UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

# Trabajo fin de grado

## Estudio y análisis del criptoanálisis lineal y diferencial: técnicas y herramientas


Sergio Galán Martín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11


Campus Internacional excelencia UAM CSIC

www.uam.es

**UNIVERSIDAD AUTÓNOMA DE MADRID**
**ESCUELA POLITÉCNICA SUPERIOR**


**Doble Grado en Ingeniería informática y Matemáticas**

# TRABAJO FIN DE GRADO

## Estudio y análisis del criptoanálisis lineal y diferencial: técnicas y herramientas

**Autor: Sergio Galán Martín**
**Tutor: Francisco de Borja Rodríguez Ortiz**

**junio 2021**

**Sergio Galán Martín**
**Estudio y análisis del criptoanálisis lineal y diferencial: técnicas y herramientas**

**Sergio Galán Martín**
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

# Agradecimientos

A mi familia, por haber estado ahí en todo momento desde el comienzo de mi andadura.

A mi tutor, por haberme ayudado a enfocar este trabajo y resolver mis numerosas dudas.

A mis compañeros, por haberme ayudado a sobrellevar la realización de este trabajo en las circunstancias que estamos viviendo.

# Resumen

Este objetivo principal de este Trabajo Fin de Grado es el estudio de dos herramientas comúnmente usadas a la hora de criptoanalizar y estudiar la robustez de algoritmos de cifrado por bloques, el criptoanálisis diferencial y el criptoanálisis lineal. Además, realizamos también una serie de experimentos en los que tratamos de estudiar la fortaleza de algunos algoritmos de cifrado entrenando diferentes modelos de aprendizaje automático para que traten de emular su funcionamiento o para detectar alguna característica de los textos cifrados con estos algoritmos.

El estudio del criptoanálisis diferencial está basado en el artículo *Differential cryptanalysis of DES-like cryptosystems* de Eli Biham y Adi Shamir, mientras que el estudio del criptoanálisis lineal está basado en el artículo *Linear Cryptanalysis Method for DES Cipher* de Mitsuru Matsui. Profundizamos en los fundamentos matemáticos de estos ataques, realizando la implementación de ambos en Python y concluyendo con un estudio de la complejidad de la realización de estos ataques en términos de los textos planos y cifrados necesarios para poder obtener la clave de cifrado.

Para los experimentos del trabajo hemos tenido como objeto de estudio tres algoritmos de cifrado. El primero es el antiguo estándar simétrico, el DES, ya que a pesar de ser un algoritmo que a día de hoy ya no es seguro (aunque una variante, el Triple DES, se sigue usando) sigue siendo interesante por haber sido el primer cifrado simétrico moderno. El segundo es el actual estándar simétrico, el AES. Es extremadamente robusto y es el algoritmo simétrico de cifrado por bloques establecido por el NIST. El último es el Speck, uno de los algoritmos de cifrado de tipo *lightweight* (pensado para ejecutarse en hardware de pocos recursos) diseñado por la NSA. Hemos usado diferentes modelos de redes neuronales artificiales para evaluar y comparar la seguridad que nos ofrecen estos tres algoritmos.

En resumen, en este trabajo hemos estudiado el criptoanálisis de algunos algoritmos de cifrado simétricos por bloques modernos desde su origen con los artículos de Biham, Shamir y Matsui hasta llegar a los enfoques más recientes en el criptoanálisis, en los que se aplican técnicas de aprendizaje automático para evaluar la fortaleza de este tipo de algoritmos de cifrado de manera directa (intentando aprender características de los textos cifrados, emular el comportamiento del cifrado o del descifrado del algoritmo, etc.) o usadas en combinación con ataques tradicionales mejorando su efectividad.

# Palabras clave

Criptoanálisis de Biham y Shamir, criptoanálisis de Matsui, distinguidores diferenciales, fortaleza de criptosistemas, AES, DES, Speck, aprendizaje automático en criptografía

# Abstract

The main objective of this Bachelor Thesis consists of the study of two tools which are commonly used when cryptanalyzing and evaluating the robustness of block cipher algorithms, differential cryptanalysis and linear cryptanalysis. In addition to that, we carry out a series of experiments in which we try to study the strength of some ciphers by training different machine learning models that try to emulate their behaviour or to detect some pattern on the ciphertexts generated by these ciphers.

The study of the differential cryptanalysis is based on the paper *Differential cryptanalysis of DES-like cryptosystems* by Eli Biham and Adi Shamir, while the study of the linear cryptanalysis is based on the paper *Linear Cryptanalysis Method for DES Cipher* by Mitsuru Matsui. We deepen the mathematical foundations of these attacks, implementing both of them in Python and concluding with a study of the complexity of the realization of these attacks in terms of the required number of plaintexts and ciphertexts to obtain the cipher key.

For the experiments we have studied three cipher algorithms. The first one is the old symmetric standard, DES. Although nowadays it is an unsafe algorithm (although a variant, the Triple DES, is still being used), it is still interesting since it has been the first modern symmetric cipher. The second one is the current symmetric standard, AES. It is extremely robust and it is the standard symmetric block cipher established by the NIST. The last one is Speck, a lightweight cipher (a cipher algorithm designed to be executed on low-end hardware) designed by the NSA. We have trained different models of neural networks in order to evaluate and compare the security these three algorithms offer us.

To sum up, in this piece of work we have studied the cryptanalysis of modern symmetric ciphers from their conception in the papers by Biham, Shamir and Matsui to the most recent approaches in cryptanalysis, in which machine learning techniques are used to evaluate the robustness of ciphers directly (trying to learn patterns of ciphertexts, emulating the behaviour of the cipher, etc.) or used in combination with traditional attacks, improving their effectiveness.

# Keywords

Biham-Shamir's cryptanalysis, Matsui's cryptanalysis, differential distinguishers, strength of cryptosystems, AES, DES, Speck, machine learning-aided cryptanalysis

# Índice

# Listas

## Lista de ecuaciones



## Lista de figuras

# Lista de tablas

# 1

# Introducción

Podemos considerar que la historia moderna de los cifrados simétricos comienza en 1976 con la estandarización por parte del National Institute of Standards and Technology (NIST) del algoritmo de cifrado Data Encryption Standard (DES) desarrollado por International Business Machines (IBM) con la participación de personalidades como Horst Feistel y Don Coppersmith. Desde entonces se han desarrollado numerosos cifrados simétricos variando los enfoques, pero los principales y más importantes tienen dos cosas en común: su estructura está basada en redes de Feistel [1] o en Substitution-Permutation Networks (SPN) [2] y tienen unas componentes no lineales llamadas **S-Boxes** o **cajas de sustitución**. Por ello, de cara a evaluar la seguridad que ofrecen este tipo de cifrados es interesante diseñar ataques que se puedan llevar a cabo en algoritmos de cifrado por bloques que cumplan estas características y que permitan evaluar de manera unificada la seguridad que nos ofrecen dichos algoritmos.

A finales de la década de los 80 Adi Shamir y Eli Biham [3] diseñaron un ataque para romper cifrados que tuvieran estas estructuras, y lo aplicaron a varios cifrados de la época descubriendo que muchos de ellos eran débiles al mismo. Durante su estudio determinaron que el DES era sorprendentemente resistente. Este ataque se conoce como **criptoanálisis diferencial** [4] y se basa en la dispar distribución de probabilidad de las diferencias entre parejas de ciertos datos en las operaciones de cifrado de dos textos planos diferentes.

En 1993 Mitsuru Masui [5] ideó un algoritmo para intentar obtener la clave de algoritmos de cifrado simétricos analizando la distribución de probabilidad de diferentes relaciones lineales o afines entre los bits de entrada y salida de cada una de las S-Boxes. En general, las S-Boxes tienen entradas y salidas de pequeño tamaño, lo que nos permite obtener esta distribución de probabilidad calculando todas las posibles combinaciones lineales. Tras ello, podremos combinar esta expresión con el resto de los elementos del cifrado para poder obtener la clave de cifrado con un cierto número de parejas de texto plano y cifrado. Este ataque se conoce como **criptoanálisis lineal**.

Por último, y dados los grandes avances que se han hecho en los últimos años en el campo del **aprendizaje automático**, se han usado este tipo de técnicas para tratar de analizar la robustez y la fortaleza de diferentes cifrados (tanto simétricos como asimétricos), obteniendo buenos resultados,

especialmente cuando se aplican en **versiones simplificadas de algoritmos simétricos** ya establecidos (menos rondas, funciones más simples, etc.) [6] [7] o **nuevos algoritmos** con funciones de cifrado y descifrado poco costosas orientados a ejecutarse en hardware de pocos recursos (*lightweight cryptography*) [8] [9] [10].

## 1.1. Motivación y objetivos

Cada día intercambiamos información con una gran cantidad de sistemas, siendo ésta altamente sensible en algunas ocasiones (credenciales, información personal, etc.). Por ello se hace necesario proteger dichos datos. La criptografía, en sus diversas formas, nos brinda una multitud de algoritmos y protocolos que nos permiten establecer una comunicación segura de forma que terceras personas que intercepten la comunicación no puedan conocer la información transmitida. Sin embargo, no todos los algoritmos criptográficos proporcionan los mismos niveles de seguridad. Una manera de cuantificar este nivel y poder comparar diferentes algoritmos criptográficos consiste en comprobar la fortaleza del cifrado frente a diferentes ataques. Éstos **pueden ser o no practicables de manera efectiva** (si no lo son, podemos hacer una estimación la complejidad teórica de dicho ataque), en cualquiera de los casos nos dan una cota de la robustez del cifrado.

Esto motiva los dos objetivos principales de este trabajo. El primero es estudiar el **estado del arte del criptoanálisis**, implementando los dos ataques clásicos que se siguen utilizando para evaluar la seguridad de nuevos algoritmos de cifrado, como son el criptoanálisis lineal y el diferencial. Estos ataques son de caja blanca, es decir, suponen conocido el funcionamiento interno del algoritmo de cifrado. Para realizar dichas implementaciones hemos elegido el DES. Podemos dividir este objetivo en varios hitos:

- Implementación del criptoanálisis diferencial en el DES con diferente número de rondas.
- Implementación del criptoanálisis lineal en el DES con diferente número de rondas.
- Exposición de los resultados del ataque (complejidad del ataque, número de parejas de textos planos y cifrados necesarias).
- Conclusiones finales de estos dos ataques.

El segundo es diseñar y realizar una serie de experimentos que nos permitan discernir cómo las **redes neuronales artificiales** nos pueden servir para **evaluar la robustez** de cualquier cifrado por bloques sin conocer el funcionamiento interno del mismo, o si pueden **mejorar** de alguna forma el criptoanálisis lineal o diferencial ya mencionados. Para ello haremos aprender diferentes modelos de redes neuronales con diferentes características de los cifrados. Podemos dividir este objetivo en varios hitos:

- Configuración del framework Keras con el backend de Tensorflow para su correcto aprovechamiento de la GPU local en el entrenamiento de redes neuronales.

- Desarrollo de una serie de experimentos en los que aplicaremos las redes neuronales a diferentes características de los algoritmos de cifrado para comprobar si nos pueden ser útiles de cara a evaluar la fortaleza de diferentes algoritmos de cifrado.

  Emulación del funcionamiento de los algoritmos de cifrado para versiones de diferente número de rondas.

  Emulación del funcionamiento de los algoritmos de cifrado ronda a ronda.

  Detección de diferenciales en el DES.

  Evaluación de la seguridad que aportan las diferentes funciones de la ronda del Advanced Encryption Standard (AES) .

- Ejecución y extracción de conclusiones de cada experimento.

Por último, extraeremos una serie de conclusiones globales del trabajo, además de proponer diferentes líneas de investigación por las que se podría expandir este proyecto.

# 2

# Estado del arte

En este capítulo estudiaremos el estado del arte del criptoanálisis de algoritmos de cifrado simétricos. Comenzaremos con una breve clasificación de los diferentes tipos de ataque, continuaremos con un estudio de la base teórica de los criptoanálisis lineal y diferencial, y concluiremos con una visión de los avances más recientes en el uso conjunto del criptoanálisis y el aprendizaje automático.

## 2.1. Tipos de ataques

Comenzamos este capítulo con una breve introducción a los tipos de ataques a algoritmos de cifrado, clasificándolos atendiendo a la información requerida para poder llevarlos a cabo.

- **Ataque de texto cifrado conocido** o Ciphertext Only Attack (COA) : Solo es necesario conocer un cierto número de textos cifrados.
- **Ataque de texto plano conocido** o Known Plaintext Attack (KPA) : Es necesario conocer cierto número de parejas de textos planos con sus correspondientes textos cifrados.
- **Ataque de texto cifrado escogido** o Chosen Ciphertext Attack (CCA) : Es necesario conocer un cierto número de parejas de textos planos con sus correspondientes textos cifrados, pero el atacante puede además elegir los textos cifrados de los que obtener estas parejas. Podemos ver este ataque como que el atacante dispone de un **oráculo** al que proporciona textos cifrados de su elección y recibe sus correspondientes textos planos.
- **Ataque de texto plano escogido** o Chosen Plaintext Attack (CPA) : Es necesario conocer un cierto número de parejas de textos planos con sus correspondientes textos cifrados, pero el atacante puede además elegir los textos planos de los que obtener estas parejas. Podemos ver este ataque como que el atacante dispone de un **oráculo** al que proporciona textos planos de su elección y recibe sus correspondientes textos cifrados.

## 2.2. Criptoanálisis diferencial

Como ya hemos mencionado, el **criptoanálisis diferencial** desarrollado por Biham y Shamir es uno de los primeros ataques desarrollados a lo que consideramos como criptografía moderna (tras la estandarización del DES en 1976). Pese a que podamos considerarlo bastante antiguo debido a la rápidez de la evolución de la informática durante las últimas décadas, este esquema se ha seguido

utilizando para evaluar la seguridad de nuevos algoritmos de cifrado [11] [12].

Con el objetivo de ilustrar los fundamentos de este ataque hemos elegido analizar el caso concreto del DES. Es importante destacar que el DES de 16 rondas es **especialmente resistente a este ataque**, ya que requiere de $2^{58}$ textos planos con sus correspondientes textos cifrados, lo que supera el coste necesario para romper este algoritmo por fuerza bruta ($2^{56}$ operaciones de cifrado). En [13] consiguieron mejorar significativamente la complejidad del algoritmo para el DES de 16 rondas, pero no estudiaremos dicha mejora en este proyecto. Don Coppersmith desveló, tras la publicación del ataque por Biham y Shamir, que el equipo de desarrollo del DES **ya conocía este ataque** y diseñaron las S-Boxes de manera que ofrecieran una gran fortaleza frente al mismo [14].

En el anexo A se dispone de una exposición detallada de los componentes internos del DES y de su funcionamiento.

### 2.2.1. Base teórica

En este apartado seguiremos los artículos de Biham y Shamir [3] [15] y utilizaremos la siguiente notación inspirada en los mismos:

- Un número en hexadecimal $n$ se denotará con $n_x$.
- $P$ representa un texto plano (64 bits) y $T$ representa su corrrespondiente texto cifrado (64 bits).
- La mitad izquierda del texto plano $P$ se denotará con $L$ y la derecha con $R$. La mitad izquierda del texto cifrado $T$ (es decir, el texto tras aplicarle el número de rondas que hayamos seleccionado) se denotará con $l$ y la derecha con $r$. $L_i$ y $R_i$ denotarán las partes izquierda y derecha (cada una de 32 bits) tras aplicar $i$ funciones de ronda. $L_0 = L$, $R_0 = R$.
- Los 32 bits de entrada a la función F a cada ronda se denotarán con $a$ (para la ronda 1) ... $d$ (para la ronda 4), donde $a$ coincide con $R$. Solo nombramos hasta dicha ronda ya que serán las que necesitemos para estudiar el criptoanálisis diferencial hasta 6 rondas.
- Los 32 bits de salida de la función F a cada ronda se denotarán con $A$ (para la ronda 1) ... $D$ (para la ronda 4).
- $K$ será la clave de cifrado (64 bits), $K_i$ la subclave de la ronda $i$ (48 bits) y $K_{ij}$ el segmento de la subclave $i$ correspondiente a la S-Box $j$ (6 bits).
- $P^{-1}(X)$ (32 bits) representa el resultado de aplicar a $X$ (32 bits) la función inversa de la permutación final $P(X)$ de la función $F$.
- $E(X)$ (48 bits) representa el resultado de aplicar a $X$ la expansión inicial de la función $F$.
- Para cualquier valor $X$ del cifrado (por ejemplo $a, L, P$ o $r$), $X'$ será $X_1 \oplus X_2$ donde $X_1$ y $X_2$ son los valores concretos de $X$ en dos ejecuciones del cifrado para dos textos planos diferentes.

Con esta notación, la función de ronda del DES se puede expresar como ($\oplus$ representa la operacion lógica Exclusive Or (XOR) bit a bit):

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus F(R_{i-1}, K_i) \end{aligned} \tag{2.1}$$

La idea de este ataque es encontrar **diferenciales** (tuplas de la forma ($P'$,$T'$)) con alta probabilidad de ocurrencia que nos ayuden a seguir la evolución de estas diferencias por las rondas del algoritmo. Por ello, lo que usaremos serán secuencias formadas por diferenciales de cada ronda que "encajen", es decir, que la diferencia de la salida de la ronda anterior coincida con la diferencia de entrada de la ronda siguiente, siempre comenzando por el diferencial de los textos planos $P'$ y concluyendo con el diferencial de los textos cifrados $T'$. Denominaremos a estas secuencias **características diferenciales de N rondas**, según el número de rondas que abarquen. Por ejemplo, una característica de 3 rondas podría ser ($1_x$, $2_x$, $3_x$, $4_x$) donde tendríamos que el XOR de los dos textos planos (cada uno de 64 bits) cumpliría que $P' = 1_x$, tras ejecutar la primera ronda tendríamos que el XOR entre ambos valores sería $(a', A') = 2_x$, de igual manera en la tercera ronda tendríamos que $(b', B') = 3_x$ y, por último, el XOR entre los dos textos cifrados sería $T' = 4_x$.

A las parejas de textos planos $(P_1, P_2)$ que cumplan una determinada característica diferencial (es decir, que sus XOR a lo largo de cada ronda del algoritmo coincidan con su correspondiente elemento de la característica diferencial) las llamaremos **parejas correctas**. Tanto para hallar características diferenciales que ocurran con alta probabilidad como para obtener bits de las subclaves nos centraremos en analizar la distribución de probabilidad de los diferenciales de las S-Boxes, ya que el resto de transformaciones del DES (expansión, permutación, etc.) son lineales, y por tanto preservan las diferencias (aunque cambien el orden de los bits). Este *script* (`DES_dif_sboxes.py`) que hemos desarrollado para este trabajo nos calcula, para cada S-Box, por un lado el número de ocurrencias de cada pareja de diferencias de entrada y salida para cada S-Box probando las $64 \cdot 64 = 2^{12}$ parejas de entradas, y por otro lado, los valores posibles de las entradas que dar lugar a cada diferencia de salida.

| **S-Box 1** | $0_x$ | $1_x$ | $2_x$ | $3_x$ | $4_x$ | $5_x$ | $6_x$ | $7_x$ | $8_x$ | $9_x$ | $A_x$ | $B_x$ | $C_x$ | $D_x$ | $E_x$ | $F_x$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $0_x$ | 64 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $1_x$ | 0 | 0 | 0 | 6 | 0 | 2 | 4 | 4 | 0 | 10 | 12 | 4 | 10 | 6 | 2 | 4 |
| $2_x$ | 0 | 0 | 0 | 8 | 0 | 4 | 4 | 4 | 0 | 6 | 8 | 6 | 12 | 6 | 4 | 2 |
| $3_x$ | 14 | 4 | 2 | 2 | 10 | 6 | 4 | 2 | 6 | 4 | 4 | 0 | 2 | 2 | 2 | 0 |
| $4_x$ | 0 | 0 | 0 | 6 | 0 | 10 | 10 | 6 | 0 | 4 | 6 | 4 | 2 | 8 | 6 | 2 |
| ... | | | | | | | | | | | | | | | | |
| $3E_x$ | 4 | 8 | 2 | 2 | 2 | 4 | 4 | 14 | 4 | 2 | 0 | 2 | 0 | 8 | 4 | 4 |
| $3F_x$ | 4 | 8 | 4 | 2 | 4 | 0 | 2 | 4 | 4 | 2 | 4 | 8 | 8 | 6 | 2 | 2 |

**Tabla 2.1:** Número de ocurrencias de cada pareja de diferencial de entrada y salida para la S-Box 1. La fila indica el diferencial de entrada y la columna el de salida

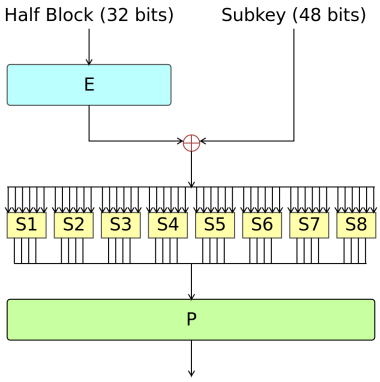



**Figura 2.1:** Función $F$ DES [1]

| Salida | Posibles entradas |
|---|---|
| $0_x$ | |
| $1_x$ | |
| ... | |
| $9_x$ | $(8_x, C_x)$ $(22_x, 26_x)$ |
| ... | |

**Tabla 2.2:** Posibles parejas de entradas para el diferencial de entrada $4_x$ según el diferencial de salida en la S-Box 1

Como podemos ver en la tabla 2.1 extraida de la primera parte de la salida del programa, tenemos que la pareja de diferencias $1_x \rightarrow 3_x$ ocurre 6 de cada 64 veces, mientras que la pareja $1_x \rightarrow 8_x$ no ocurre nunca o que la pareja $1_x \rightarrow A_x$ ocurre 12 de cada 64 veces. Esto nos indica que la distribución de probabilidades de los diferenciales de salida es **altamente desigual**, y es lo que aprovecharemos para poder criptoanalizar el cifrado. En la tabla 2.2 (se añade la figura 2.1 para facilitar la interpretación de los datos de la tabla) podemos ver los resultados de la segunda parte del programa (la correspondiente a la búsqueda de las parejas de texto plano que dan lugar a cada diferencial de salida) para el diferencial de entrada $4_x$ en la primera S-Box. De dicha tabla podemos deducir que, si en la primera S-Box se tiene que la diferencia en bits de las dos entradas es $4_x$ y la de las dos salidas es $9_x$, entonces sabemos que la pareja de entradas que se le ha suministrado a la S-Box tiene que ser o bien $(8_x, C_x)$ o bien $(22_x, 26_x)$ (ya que son las dos únicas parejas de entradas cuya diferencia es $4_x$ y cuya salida por la S-Box 1 tiene como diferencia $9_x$).

Veremos ahora cómo esa información nos ayuda a obtener la clave si podemos cifrar un determinado número de textos planos **elegidos por el criptoanalista** (por tanto este es un CPA). Con el fin de ilustrar esta idea de manera sencilla vamos a trabajar sobre una versión del DES omitiendo la permutación inicial y final del cifrado (ya que, como comentamos anteriormente, no aporta ningún tipo de robustez adicional frente a este ataque al ser una reordenación de los bits de la entrada y de la salida).

Ya mencionamos que tanto la expansión inicial como la permutación final de la función $F$ del DES son lineales, sus diferencias también lo son, por lo que para estudiar el comportamiento global de los diferenciales del algoritmo de cifrado nos bastará con analizar las diferencias de entrada y salida de las S-Boxes. También es importante destacar que la diferencia de dos datos a la entrada (el XOR entre ambos) de una S-Box **no depende de la clave** como podemos apreciar en la siguiente ecuación, donde vemos este hecho en el caso concreto de la función $F$ en la primera ronda del cifrado:

$$(E(a_1) \oplus K_1) \oplus (E(a_2) \oplus K_1) = E(a_1) \oplus E(a_2) = E(a_1 \oplus a_2) = E(a'). \tag{2.2}$$

[1] Extraido de https://en.wikipedia.org/wiki/Data_Encryption_Standard

En el capítulo 3 aplicaremos estos resultados teóricos realizando una implementación propia del ataque en Python para el DES con diferente número de rondas.

## 2.3. Criptoanálisis lineal

Este ataque fue diseñado por Mitsuru Matsui en 1993 [5], ofreciendo unos mejores resultados frente al DES de 16 rondas que el diferencial de Biham y Shamir, ya que son necesarios unos $2^{46,5}$ textos planos con sus correspondientes textos cifrados frente a los $2^{58}$ del diferencial. A diferencia de este, el criptoanálisis lineal no necesita parejas de textos planos y cifrados a los que calcular la diferencia de sus bits en diferentes momentos del algoritmo, sino que trabaja con varios textos planos y sus correspondientes textos cifrados de manera independiente. Además, en vez de fundamentarse en la dispar distribución de probabilidad de los diferenciales, este ataque se basa en la probabilidad (distinta de 1/2) de que se cumpla cierta expresión lineal de los bits del texto plano, del texto cifrado y de las subclaves de cifrado. Veremos cómo aprovechar estos sesgos probabilísticos para poder obtener la clave en la sección 3.2.

Al igual que el criptoanálisis diferencial, a pesar a ser un ataque relativamente antiguo, se ha seguido usando para evaluar la robustez y fortaleza de nuevos algoritmos de cifrado [16] [17]. En la siguiente sección analizaremos los fundamentos de este ataque en el caso concreto del algoritmo de cifrado DES.

### 2.3.1. Base teórica

Como ya hemos mencionado, este ataque trata de buscar **relaciones lineales** entre los bits del texto plano, del texto cifrado y de la clave usada para cifrar que se cumplan con una probabilidad superior al 50 %. Ya que la única parte no lineal en los cifrados simétricos son las S-Boxes, buscaremos relaciones que se cumplan en ellas, ya que el resto de los componentes tiene un comportamiento lineal que se cumple en el 100 % de los casos. Para conocer la probabilidad de que se cumpla la relación construida en base a las obtenidas de las S-Boxes necesitamos introducir algunos conceptos previos.

Procedemos a enunciar el **lema del apilamiento** de Matsui [5]. Sean $X_1, ..., X_n$ variables aleatorias que toman los valores 0 y 1 con probabilidades $1/2 + \epsilon_i$ y $1/2 - \epsilon_i$ respectivamente, donde $-1/2 \leq \epsilon_i \leq 1/2$ es lo que denominaremos el **sesgo** de dicha variable aleatoria. Entonces la probabilidad de que la variable aleatoria $X_1 \oplus ... \oplus X_n$ sea igual a 0 viene dada por la siguiente ecuación:

$$Pr(X_1 \oplus ... \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \epsilon_i. \tag{2.3}$$

En otras palabras, esto nos asegura que el sesgo de la variable aleatoria $X_1 \oplus ... \oplus X_n$ es $2^{n-1} \prod_{i=1}^{n} \epsilon_i = \epsilon_{1,...,n}$. Podemos apreciar que si alguna de las variables aleatorias tiene sesgo 0, entonces el sesgo de la suma de todas ellas será 0, y por tanto tomará los valores 0 y 1 con igual probabilidad. Además, gracias a este lema y a las propiedades de la función XOR, tenemos una manera de combinar los sesgos de dos expresiones lineales de manera sencilla. Por ejemplo, si tenemos que $Z_1 = X_1 \oplus X_2 \oplus X_3$ tiene sesgo $\epsilon_{1,2,3}$ y $Z_2 = X_1 \oplus X_4$ tiene sesgo $\epsilon_{1,4}$, entonces sabemos que $Z_1 \oplus Z_2 = X_2 \oplus X_3 \oplus X_4$ tiene sesgo $2 \cdot \epsilon_{1,2,3} \cdot \epsilon_{1,4}$. Es importante recalcar que un elemento lineal del cifrado se puede ver como una variable aleatoria con sesgo $1/2$, por lo que combinarlo con otro elemento (por ejemplo, una S-Box) **no modifica el sesgo** (multiplicaríamos el sesgo por $1/2$ y lo dividiríamos por $2$). Ese hecho será el que aprovecharemos para obtener una expresión lineal (sesgada) del comportamiento global del DES.

Para encontrar estas expresiones hemos implementado un *script* (`DES_lin_sboxes.py`). Este *script* comprueba, para cada S-Box (8 S-Boxes) y para cada expresión lineal que involucra bits de la entrada de la S-Box (6 bits) y bits de la salida de la S-Box (4 bits), el número de entradas que hacen que dicha expresión lineal se cumpla, obteniendo la expresión lineal con más sesgo (**en módulo**) de cada una de las S-Boxes. La salida de dicho programa se puede ver en la figura 2.2.

```
S-box 1 -> Expresión: 0100001111    Sesgo: -18/64
S-box 2 -> Expresión: 1000101011    Sesgo: -16/64
S-box 3 -> Expresión: 1000101111    Sesgo: 16/64
S-box 4 -> Expresión: 1000101111    Sesgo: -16/64
S-box 5 -> Expresión: 0100001111    Sesgo: -20/64
S-box 6 -> Expresión: 0100000111    Sesgo: -14/64
S-box 7 -> Expresión: 1110110100    Sesgo: -18/64
S-box 8 -> Expresión: 0100001111    Sesgo: -16/64
```

**Figura 2.2:** Salida del programa que calcula la expresión lineal más sesgada de cada S-Box del DES

Los bits de la expresión de la salida del programa representan, de izquierda a derecha, los 6 bits de la entrada (numerados de 0 a 5) y los 4 bits de la salida (numerados de 0 a 3), indicando un 1 que dicho bit pertenece a la expresión y 0 lo contrario. Por ejemplo, la salida del programa nos indica que la mejor expresión lineal (con un sesgo de -20/64) para la quinta S-Box es:

$$X[1] \oplus Y[0] \oplus Y[1] \oplus Y[2] \oplus Y[3] = 0. \tag{2.4}$$

Donde $X[i]$ es el i-ésimo bit de la entrada a la S-Box y $Y[j]$ es el j-ésimo bit de la salida de la S-Box.

En el capítulo 3 veremos cómo usar esta teoría y las expresiones lineales de las S-Boxes obtenidas por el programa para obtener la clave de cifrado teniendo una cierta cantidad de textos planos y sus correspondientes textos cifrados.

## 2.4. Criptoanálisis con aprendizaje automático

El rápido avance de las técnicas de aprendizaje automático en los últimos años ha hecho que surjan aplicaciones de esta disciplina en innumerables ámbitos tanto dentro como fuera de la informática. Por ello no es extraño que haya llegado hasta el mundo del criptoanálisis. La aplicación de esta disciplina al estudio de la robustez de algoritmos de cifrado ha tenido tanto partidarios como detractores. Por ejemplo, en la sección 1 del capítulo 7 de [18] Schneier afirma que "*Neural networks work well in structured environments where there is something to learn, but not in the high-entropy, seemingly random world of cryptography.*". Sin embargo, a día de hoy hay numerosos artículos que describen mejoras de los criptoanálisis tradicionales de varios algoritmos de cifrado, dejando patente que el aprendizaje automático se puede aplicar a las técnicas tradicionales de criptoanálisis o como metodología complementaria para probar la fortaleza de nuevos algoritmos que surjan.

Por poner algún ejemplo, en [7], usando redes neuronales artificiales, los autores consiguen obtener más de la mitad de los bytes de texto plano a partir del texto cifrado usando diferentes versiones de AES con una probabilidad de más del 63 % (esto, obviamente, no criptoanaliza el AES de manera completa, pero da una idea de la fortaleza del algoritmo). En [19] se usan diferentes técnicas de aprendizaje automático (perceptrón multicapa, Recurrent Neural Network (RNN) y Long Short-Term Memory (LSTM) ) para generar **distinguidores diferenciales** de diferentes *lightweight ciphers*, que son características diferenciales que permiten distinguir con alta probabilidad entre textos cifrados con un algoritmo de cifrado y cadenas de bits generadas de manera aleatoria. Estos distinguidores sirven para evaluar la fortaleza de algoritmos de cifrado, ya que cuanto más difícil sea distinguir la salida de un cifrado de bits aleatorios, más robusto será el mismo. En particular son importantes los resultados obtenidos en el criptoanálisis al algoritmo de cifrado Speck [9] [10] surgidos tras el artículo de Gohr [8]. Nuestro enfoque en este aspecto será algo más general, tratando de entrenar redes neuronales para aprender ciertas características de los textos cifrados (o los valores intermedios del cifrado), aunque también realizaremos un análisis de la fortaleza de las funciones internas de la función de ronda del AES.

# 3

# Desarrollo

En este capítulo procederemos a implementar el criptoanálisis diferencial y el lineal en el caso concreto del DES basándonos en los conceptos descritos en el capítulo 2, además de introducir la metodología y los experimentos que llevaremos a cabo para estudiar cómo el aprendizaje automático puede ayudarnos a evaluar la robustez de diferentes algoritmos de cifrado.

## 3.1. Criptoanálisis diferencial

Ya que el funcionamiento del criptoanálisis diferencial del DES varía bastante en función del número de rondas, hemos optado por implementar las versiones de 1, 4 y 6 rondas con el objetivo de ilustrar los mecanismos que van apareciendo al aumentar el número de rondas.

### 3.1.1. 1 Ronda

En la versión de una única ronda de DES conocemos todos los bits de entrada y de salida de la función $F$ si disponemos de un texto plano y su correspondiente texto cifrado. Con esos datos (y recordando la ecuación 2.1 de ronda y la figura 2.1 que ilustra la función $F$) disponemos también de los bits de salida de cada S-Box aplicando la permutación inversa $P^{-1}$ dentro de la función $F$ y los bits de entrada de cada S-Box aplicando la expansión $E$, estos últimos a falta de operarlos mediante un XOR con los bits de la subclave correspondiente a la primera ronda del cifrado. Las S-Boxes del DES cumplen ciertos criterios de diseño especificados en [14], entre los cuales se encuentra el hecho de que en cada fila de la S-Box aparecen los números del 0 al 15 sin repetición, por lo que si conocemos la salida de la S-Box, entonces la entrada tiene que ser alguna de las cuatro (una por fila, ya que en una fila siempre se encuentran las 16 posibles salidas) correspondientes a dicha salida. Por ejemplo, para la S-Box 1 y la salida $6_x$ tenemos que las cuatro posibles entradas son $001010_2, 001001_2, 100010_2$ y $100111_2$, que son los posibles valores de los seis primeros bits de $E(R) \oplus K_1$. Por tanto, si ahora calculamos los valores $E(R) \oplus Z$ siendo $Z$ cada uno de los cuatro valores obtenidos, tendremos los cuatro únicos valores posibles para los seis primeros bits de la subclave $K_1$. Si repetimos esto para el resto de las S-Boxes habremos conseguido, con una única pareja de texto plano y su correspondiente

texto cifrado, 4 posibilidades para cada bloque de 6 bits de la subclave $K_1$.

Repitiendo este procedimiento para más parejas de textos planos y cifrados iremos reduciendo el conjunto de posibles valores de cada bloque de 6 bits a una única posibilidad. En ese momento habremos hallado todos los bits de la subclave, tras lo cual desharemos el *key schedule* del DES (que es sencillo de invertir ya que tan solo consta de rotaciones y permutaciones, para más detalles sobre esta función consultar el Anexo A). Una vez hecho esto tendremos la clave de cifrado usada (realmente solo nos dará 48 bits de la clave ya que el resto no se usan en la primera ronda, y por tanto en esta versión del DES hay varias claves que generan la misma subclave y nos serán indistiguibles). Es importante destacar que en este caso **no necesitamos escoger los textos planos y cifrados de ninguna manera concreta**, nos valen textos planos aleatorios, por lo que se trata de un KPA (que ya definimos en 2.1). La implementación de este algoritmo en la versión de una ronda del DES se encuentra en el fichero `DES_dif_1R.py`.

### 3.1.2. 4 Rondas

En este caso no podemos conocer directamente los bits a la salida de la función $F$ en la última ronda, por lo que nos es necesario empezar a trabajar con las características diferenciales que hemos mencionado en el capítulo anterior. Usaremos la característica diferencial de una ronda (según se muestra en la figura 3.1) que se cumple con una probabilidad $p = 1$, esto es, que si introducimos dos textos planos que difieran solo en su tercer bit, entonces la salida de ambos en un DES de una ronda diferirá también únicamente en su tercer bit, ya que al ser $R' = 00000000_x$ entonces $a'$ será $00000000_x$, y por tanto $A'$ también lo será.

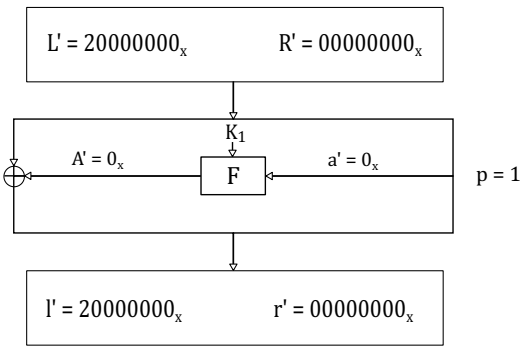



**Figura 3.1:** Característica diferencial de 1 ronda para el criptoanálisis diferencial de 4 rondas

La razón por la que se ha seleccionado que $L'$ sea $20000000_x$ es minimizar las S-Boxes **activas** en la segunda ronda, es decir, S-Boxes con diferencias de salida distintas de 0, ya que cuantas más S-Boxes haya involucradas menor será la probabilidad de ocurrencia del diferencial, y necesitaremos más textos planos para llevar a cabo el criptoanálisis. En la segunda ronda $L' \oplus A' = L'$ es la entrada a la función $F$. De esta forma tenemos que las diferencias de salida de las S-Boxes 2 a 8 es 0. Ahora,

vemos que $D' = l' \oplus R' \oplus B'$ (donde recordemos que $B' = F(A' \oplus L', K_2)$ y $l'$ es la diferencia en bits de la parte izquierda de los textos cifrados tras las 4 rondas). Además, como $R' = 00000000_x$ y 28 bits de $B'$ (los correspondientes a los 28 últimos tras aplicarles la permutación $P$ de la función $F$) son 0, conocemos con seguridad 28 bits de $D'$, que al aplicarles la permutación inversa $P^{-1}$ nos dan las diferencias de las salidas de las S-Boxes 2 a 8 en la última ronda. Al tener esas diferencias y los bits de entrada de esas S-Boxes antes de operarlos con la subclave (son $r_1$ y $r_2$ tras aplicar la expansión $E$) procedemos a hallar la clave. Para ello calcularemos, para cada S-Box $j$ con $j$ de 2 a 8 y para los 64 posibles valores del segmento (es decir, cada uno de los grupos de 6 bits de la subclave asociados a cada S-Box) de la subclave correspondiente a dicha S-Box ($K_{4j}$), si se cumple la siguiente ecuación para los 28 últimos bits, donde $E(r_i)_j$ representa los 6 bits de $E(r_i)$ correspondientes a la S-Box $j$, y $S_j$ la función que representa la aplicación de la S-Box $j$:

$$S_j(E(r_1)_j \oplus K_{4j}) \oplus S_j(E(r_2)_j \oplus K_{4j}) = P^{-1}(D'). \tag{3.1}$$

Los valores de $K_{4j}$ que cumplan dicha ecuación serán nuestros candidatos a segmento de 6 bits de subclave correspondiente a la S-Box $j$, pudiendo descartar con seguridad los que no la cumplan. Ahora tan solo tenemos que repetir el procedimiento para más parejas de textos planos y textos cifrados hasta que quedemos con una única posibilidad para cada segmento de 6 bits de las S-Boxes 2 a 8, tras lo cual podemos deshacer el *key schedule* del DES para obtener 42 bits de los 56 bits efectivos (omitiendo los de paridad) de la clave de cifrado. Para concluir el ataque tan solo tenemos que probar a cifrar alguno de los textos planos que tenemos con las $2^{14}$ (las que nos quedan de las $2^{56}$ iniciales) posibilidades de claves hasta obtener el texto cifrado correspondiente, con lo que habremos obtenido la clave completa. El código que implementa este procedimiento se puede encontrar en el fichero `DES_dif_4R.py`.

En la figura 3.2 se puede ver la información que el programa nos devuelve al ejecutarlo, entre la que se encuentra el número de textos que ha necesitado generar el programa hasta obtener la clave.

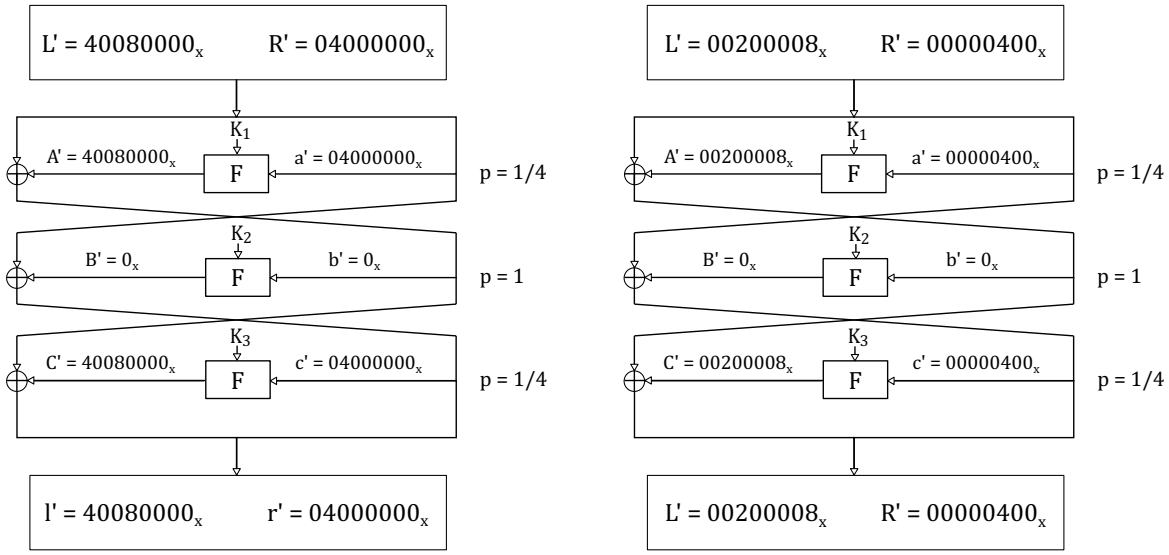
```
$ python3 DES_dif_4R.py 0x0110011001100110
Iteraciones utilizadas:  4
Clave obtenida: 0b00N00M0_LKJ1000_0000000_0001IH0_00GF00E_DC01000_B000000_00A1000_
Clave real:     0b1000100000000000100010000000000010001000000000100010000
Los _ son bits de paridad, las letras son 14 bits desconocidos
Obteniendo los 14 bits restantes por fuerza bruta
La clave de cifrado es 0x110011001100110
```

**Figura 3.2:** Salida del programa que implementa el criptoanálisis diferencial para el DES de 4 rondas

### 3.1.3. 6 Rondas

Para el DES de 6 rondas ya no existe ninguna característica diferencial con 100 % de probabilidad de ocurrencia que nos ayude a obtener bits de la última subclave del cifrado, por lo que tendremos

que usar características diferenciales con una alta probabilidad de ocurrencia. Además, por la propagacion de las diferencias de bits distintos de 0 a través de las diferentes rondas, tampoco podemos tener la certeza de conocer tantas entradas a S-Boxes como en el caso anterior. Con todo ello, usaremos dos características diferenciales de tres rondas (las usadas por Biham y Shamir en [3]), cada una de las cuales nos dará, con una probabilidad de 1/16 (valor obtenido de la salida del *script* `DES_dif_sboxes.py`), las entradas de 5 S-Boxes en la última ronda, y por tanto el valor de 30 bits de la última subclave. La primera de estas características es la que se puede apreciar en la figura 3.3(a).




(a) Primera característica

(b) Segunda característica

**Figura 3.3:** Características diferenciales de 3 rondas para el criptoanálisis diferencial de 6 rondas

Con esa característica diferencial, y notando que la diferencia de entradas a 5 S-Boxes en la cuarta ronda ($d'$) es 0, tenemos que su salida ($D'$) también lo es. Siguiendo hasta la sexta ronda tenemos que $F' = D' \oplus l' \oplus c'$, de los que conocemos 30 bits con una probabilidad de 1/16. Como ya hemos visto, las diferencias de los bits de entrada a las S-Boxes coinciden con la diferencia entre las partes derechas de los textos cifrados ($r'$). Ahora tenemos que generar un número suficiente de parejas de textos planos y sus correspondientes cifrados (en este caso no es suficiente con ir descartando, ya que al darse la característica diferencial con una probabilidad menor que 1 habrá casos en los que los bits de la clave correcta no sean obtenidos como posibles valores de dichos bits). Tras ello, obtenemos los posibles valores de los 30 bits de la última subclave correspondientes a las S-Boxes 2, 5, 6, 7 y 8 de manera idéntica al caso de 4 rondas y nos quedamos con los más probables, ya que si generamos suficientes muestras (en la siguiente sección veremos cómo estimar el número de muestras necesarias) los bits correctos de la subclave serán los más sugeridos por los cálculos. De esta forma habremos obtenido 30 bits de la subclave con una alta probabilidad, que reforzaremos con la siguiente característica diferencial y que también nos proporcionará 12 bits más de la subclave.

La segunda característica diferencial que usaremos es la de la figura 3.3(b). De manera similar a la anterior, nos proporciona, con una probabilidad de 1/16, los valores de los 30 bits de la salida de las S-Boxes 1, 2, 4, 5 y 6 en la ronda 6. Siguiendo un procedimiento análogo al realizado con la característica diferencial anterior, podemos obtener con una alta probabilidad los bits de los segmentos de la subclave correspondientes a dichas S-Boxes. Ahora tan solo tenemos que comprobar que los bits de los segmentos que coinciden en ambos casos (los asociados a las S-Boxes 2, 5 y 6). Podría darse el caso de que no coincidan debido a la naturaleza probabilística del diferencial (una clave incorrecta ha sido la más sugerida por el algoritmo), aunque con el suficiente número de parejas de texto plano y cifrado es poco probable que esto ocurra. En caso de que suceda se podría proceder a probar con más parejas de textos planos y cifrados o, simplemente, volver a ejecutar el algoritmo. En cualquier caso, con esto habremos obtenido 42 bits de la subclave $K_6$, con lo que deshaciendo el algoritmo de generación de subclaves del DES obtenemos 42 bits de la clave $K$.

Para concluir el ataque procedemos de manera análoga a los casos anteriores: buscando el resto de bits mediante una búsqueda por fuerza bruta. Es importante recalcar que esta fuerza bruta para encontrar los 14 bits faltantes (probar $2^{14}$ claves) lleva un tiempo prácticamente despreciable en contraste con el que supondría el ataque puramente por fuerza bruta con $2^{56}$ claves. El código correspondiente a este ataque se puede encontrar en el fichero `DES_dif_6R.py`.

Para llevar a cabo el criptoanálisis diferencial del DES con un mayor número de rondas, digamos $N$, se usan características diferenciales de $N-3$ rondas y se obtiene cierto número de bits (que depende de las características diferenciales y del número de rondas) de la última subclave (y por tanto de la propia clave de cifrado) de manera idéntica a la llevada a cabo en el DES de 6 rondas. Sin embargo, el número de textos planos con sus correspondientes textos cifrados aumenta significativamente, ya que la probabilidad de estas características va disminuyendo conforme aumenta el número de rondas.

### 3.1.4. Complejidad y conclusiones

El criptoanálisis diferencial es un CPA (ya que necesitamos textos planos cuya diferencia en bits coincida con los diferenciales usados para el ataque) y probabilístico (para un número de rondas por encima de 3), es decir, podría darse el caso de que no fuese capaz de devolver la respuesta correcta en ciertas ocasiones, teniendo que volver a ejecutar el algoritmo para generar nuevas parejas de textos planos y textos cifrados. Por ello, es interesante preguntarnos cuántas parejas de textos planos y sus correspondientes cifrados son necesarias para que el ataque se complete satisfactoriamente con una alta probabilidad.

Para poder analizar esta probabilidad correctamente, introducimos primero el concepto de **relación señal/ruido**, el cual tiene su origen en la transmisión de ondas pero que utilizaremos con una definición adaptada al problema que tenemos entre manos. Definimos la Signal-Noise Ratio (S/N) o relación señal/ruido de un ataque como el cociente entre el **número de parejas correctas** (el número de

parejas de textos planos y cifrados que cumplen una cierta característica diferencial) y la **media de subclaves incorrectas sugeridas** por el ataque (es decir, el número medio de textos planos y cifrados de los que el algoritmo deduce de manera incorrecta los bits de la subclave que estamos obteniendo). Es importante notar que las subclaves incorrectas dependen de la **manera de contar** estas subclaves. Por ejemplo, en nuestra implementación del ataque para el DES de 6 rondas hemos usado un contador para cada una de las $2^{30}$ posibilidades de los 30 bits que podemos obtener de la subclave, pero también podríamos haberlo implementado con $5 \times 2^6$ contadores para cada una de las posibilidades de la entrada de cada una de las 5 S-Boxes. Esto tiene consecuencias en el coste temporal y en el número de textos planos y cifrados necesarios: cuantos más bits de la subclave tengamos en cuenta de manera simultánea menos textos necesitaremos, pero más memoria utilizará el programa (ya que tendrá que mantener una mayor cantidad de contadores) y viceversa. Cuantificaremos y formalizaremos esta idea en los siguientes párrafos.

Podemos obtener el número (esperado) $n$ de parejas correctas multiplicando el número de parejas generadas para el ataque por la probabilidad $p$ de ocurrencia de la característica diferencial del ataque. Para calcular el número de parejas incorrectas sugeridas necesitaremos conocer (además del número de parejas generadas $n$) el número $k$ de bits de la subclave que estamos contando (los dos métodos de conteo descritos al final del párrafo anterior corresponden a los casos $k = 30$ y $k = 6$, respectivamente) y el número medio $\gamma$ de contadores que se incrementan tras analizar una pareja. Con todo esto, podemos calcular la relación señal/ruido de un ataque usando la siguiente ecuación (extraida de [3]):

$$S/N = \frac{np}{n\frac{\gamma}{2^k}} = \frac{2^k p}{\gamma} \tag{3.2}$$

De la ecuación de la relación señal/ruido deducimos que su valor **no depende del número de parejas generadas**, lo que nos perimitirá obtener el número de parejas de textos planos y cifrados para que el algoritmo obtiene la clave correcta tan solo usando $p, k$ y $\gamma$.

Resultados experimentales de Shamir y Biham [3] establecen que si el valor de la relación señal/ruido está entre 1 y 2 necesitaríamos entre 20 y 40 parejas correctas y que para valores extremadamente altos 3 o 4 parejas correctas son suficientes. Por ejemplo, para el caso del criptoanálisis diferencial del DES de 6 rondas (en el que las características diferenciales tenían como probabilidad $p = 1/16$) contando sobre los $30$ bits simultáneamente ($k = 30$) tenemos que la relación señal/ruido es $\frac{2^{30}\frac{1}{16}}{4^5} = 2^{16}$ (donde hemos usado que cada S-Box sugiere de media 4 valores y estamos analizando 5 S-Boxes). Al ser un valor moderadamente alto, Biham y Shamir establecen que 7 u 8 parejas correctas son suficientes, por lo que al tener cada una de las características diferenciales de la figura 3.3 una probabilidad de ocurrencia de $1/16$, tenemos que necesitaríamos entre $112$ y $128 \approx 2^7$ parejas para que el ataque sea satisfactorio con una alta probabilidad.

A más rondas, la probabilidad de ocurrencia de las características diferenciales que podemos en-

contrar decrece inevitablemente, además de disminuir su relación señal/ruido, por lo que iremos necesitando generar un número mayor de parejas. En la tabla 3.1 podemos ver el número de parejas necesarias calculadas por Biham y Shamir en su trabajo [3] para cada número de rondas del DES.

| **Rondas** | 4 | 6 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Parejas** | $2^3$ | $2^7$ | $2^{15}$ | $2^{25}$ | $2^{34}$ | $2^{35}$ | $2^{42}$ | $2^{43}$ | $2^{50}$ | $2^{51}$ | $2^{57}$ |

**Tabla 3.1:** Parejas necesarias para romper el DES en función del número de rondas usando criptoanálisis diferencial

Como podemos apreciar, para todas las rondas este ataque requiere del cálculo de menos operaciones de cifrado que un ataque por fuerza bruta ($2^{56}$ operaciones de cifrado) salvo para el ataque a la versión de 16 rondas, para el cual necesitamos llevar a cabo $2 \times 2^{57} = 2^{58}$ operaciones de cifrado (además de ejecutar el propio criptoanálisis diferencial sobre esas parejas). Por lo tanto, el criptoanálisis diferencial del DES mejora el ataque por fuerza bruta para un número de rondas inferior a 16, siendo inefectivo de 16 rondas en adelante. Esto coincide con lo establecido por Don Coppersmith en [14] años después del paper de Shamir y Biham, en el que afirmaba que los creadores del DES (él mismo entre ellos) conocían este ataque y ajustaron los parámetros del estándar (número de rondas, permutación $P$, expansión $E$ y S-Boxes) para que fuese resistente al mismo.

## 3.2. Criptoanálisis lineal

En el caso del criptoanálisis lineal el procedimiento para diferente número de rondas (a partir de 4, ya que Matsui no especifica ningún tipo de ataque para el DES de menos rondas) es fácilmente generalizable, por lo que nuestra implementación admite como parámetro el número de rondas de la versión del DES que queramos atacar. Sin embargo, con el objetivo de ofrecer una exposición clara y sencilla del ataque, en este apartado nos centraremos en la versión concreta del DES de 4 rondas.

Con las herramientas que introducimos en la sección 2.3.1, podemos combinar las mejores expresiones lineales de las S-Boxes con el resto de elementos lineales del cifrado (expansión, permutación, etc.) sin modificar la probabilidad de ocurrencia de la expresión (como demostramos en dicha sección), y con otras S-Boxes perdiendo cierta probabilidad de ocurrencia indicada por el **lema del apilamiento** de Matsui. La tabla completa para las mejores expresiones lineales del DES de rondas entre 3 y 20 rondas se puede encontrar en [5], de la que extraemos que la mejor expresión del DES de 3 rondas es la indicada en la siguiente ecuación, y ocurre con una probabilidad de $1/2 + 1{,}56 \cdot 10^{-3}$ (hemos adaptado la numeración de bits a la usada en nuestra implementacion del DES y la nomenclatura a la introducida en la sección 2.2.1):

$$L[3] \oplus L[8] \oplus L[14] \oplus L[25] \oplus R[17] \oplus l[3] \oplus l[8] \oplus l[14] \oplus l[25] \oplus r[17] = K_1[26] \oplus K_3[26]. \quad (3.3)$$

Donde recordamos que $L$ y $R$ son, respectivamente, las mitades izquierda y derecha del texto plano, $l$ y $r$ son las mitades izquierda y derecha del texto cifrado, $K_i$ representa la subclave de la ronda $i$ y el número entre corchetes es el bit del elemento al que acompaña.

Con esta expresión podemos obtener varios bits de la clave de cifrado si disponemos de cierto número de parejas de texto plano y su correspondiente texto cifrado (abordaremos la cuestión de la elección de este número de parejas en la sección 3.2.1) en el DES de 4 rondas. Para ello, primero propagaremos hacia atrás los valores $l$ y $r$ una ronda (recordando la ecuación 2.1), obteniendo el equivalente a un DES de 3 rondas (usaremos una expresión del DES de 3 rondas para obtener la clave de un DES de 4 rondas ya que propagaremos hacia atrás los valores de la última ronda) donde ahora su parte izquierda toma el valor $r$ y su parte derecha toma el valor $l \oplus F(r, K_4)$. Si sustituimos estos valores en la expresión 3.3 del DES de 3 rondas, obtenemos la siguiente ecuación:

$$L[3] \oplus L[8] \oplus L[14] \oplus L[25] \oplus R[17] \oplus r[3] \oplus r[8] \oplus r[14] \oplus r[25] \oplus l[17] \oplus F(r, K_4)[17] = K_1[26] \oplus K_3[26]. \tag{3.4}$$

Ya que sabemos que dicha expresión se cumple con una probabilidad distinta de $1/2$ y para cada pareja de texto plano y cifrado conocemos el valor de la parte izquierda de expresión, podemos calcular el valor de la expresión para los $2^6$ posibles valores de la parte de la subclave $K_4$ de la S-Box correspondiente al bit 17 de dicha subclave (que es la S-Box 1), anotando en un contador las veces que cada posible valor de la parte de la subclave hace que la parte izquierda de la expresión tome el valor 0. El contador que más se aleje de la mitad del número de parejas de texto plano y texto cifrado será, con una alta probabilidad, el asociado a la parte de la subclave correcta, con lo que obtenemos 6 bits de la clave de cifrado (los de $K_{41}$, la parte de la cuarta subclave asociada a la S-Box 1). Además, también podemos saber el valor de la parte derecha de la ecuación, ya que si el contador está por encima de la mitad del número de parejas de textos planos y cifrados usados tendremos que el valor de ambos lados de la expresión es 0, mientras que será 1 en caso contrario. De esta forma hallamos un total de 7 bits de la subclave. Esta idea se plasma en el algoritmo 3.1 (algoritmo extraído de [5]).

Aplicando este algoritmo con la ecuación 3.4 conseguimos obtener 6 bits de la clave (tras deshacer el algoritmo de generación de subclaves del DES), además del valor de la expresión lineal de la derecha, lo que nos aporta un bit más de la clave.

De manera análoga, podemos propagar los valores de $L$ y $R$ a través de la primera ronda, obteniendo la siguiente expresión lineal:

$$R[3] \oplus R[8] \oplus R[14] \oplus R[25] \oplus L[17] \oplus F(R, K_1)[17] \oplus l[3] \oplus l[8] \oplus l[14] \oplus l[25] \oplus r[17] = K_2[26] \oplus K_4[26]. \tag{3.5}$$

**input :** $N$ parejas de texto plano y cifrado de un DES de $m$ rondas $P$ y una expresión lineal $e$ con probabilidad $p$
**output:** Algunos bits de la clave y el valor de la parte izquierda de la expresión $e$

```
foreach candidato K_m^(i) de K_m do contadores[i] ← CuentaCeros(K_m^(i), e, P);
Tmax ← Max( contadores );
Tmin ← Min( contadores );
if Abs( Tmax- N/2 ) > Abs( Tmin- N/2 ) then
    SKPred ← Argmax( contadores );
    if p > 1/2 then
        ExpPred ← 0;
    else
        ExpPred ← 1;
    end
else
    SKPred ← Argmin( contadores );
    if p > 1/2 then
        ExpPred ← 1;
    else
        ExpPred ← 0;
    end
end
```

**Algoritmo 3.1:** Algoritmo que permite obtener bits de la clave usando la mejor expresión lineal y parejas de textos planos y cifrados

Es importante destacar que en este caso no solo ha cambiado la parte izquierda de la ecuación, sino que las subclaves involucradas en la parte derecha de la ecuación también han cambiado. Con esta segunda expresión podemos obtener otros 7 bits de la clave siguiendo el mismo procedimiento seguido para la primera expresión. Una vez obtenidos estos 14 bits, el resto de la clave se obtiene mediante fuerza bruta, probando todas las posibles combinaciones ($2^{42}$, en contraste con las $2^{56}$ del ataque por fuerza bruta) para el resto de los bits hasta encontrar la correcta.

Nuestra implemenetación de este ataque para un número arbitrario de rondas se puede encontrar en el fichero `DES_lin.py`.

En la figura 3.4 se puede ver la información que el programa nos devuelve al ejecutarlo sobre un DES de 4 rondas. Por ejemplo, podemos apreciar los bits obtenidos de las subclaves de la primera y última ronda correspondientes a la S-Box indicada por la expresión lineal (esta S-Box nos la indican las máscaras), el valor del XOR de los bits de la parte derecha de la expresión lineal para cada uno de los dos casos y la expresión obtenida antes de obtener los bits faltantes por fuerza bruta (en esa expresión los guiones bajos representan los bits de paridad y el resto de símbolos representan bits desconocidos antes de llevar a cabo la fuerza bruta).

```
$ python3 DES_lin.py 4 0x0101010101315161
Subclave 1 obtenida: 0b0
Máscara subclave 1: 0b1111110000000000000000000000000000000000000000
Subclave 4 obtenida: 0b1000000000000000000000000000000000000000000000
Máscara subclave 4: 0b1111110000000000000000000000000000000000000000
El xor de los bits de la clave al descifrar la ronda 1 es 0b0
El xor de los bits de la clave al descifrar la ronda 4 es 0b0
nmlkjih_000gfed_0cba`^]_\[ZYXWV_U00TSRQ_P0ONMLK_0J0IHGF_ED10CBA_
0x101010101315161
Tiempo invertido en fuerza bruta:  1.5639164447784424 ms
```

**Figura 3.4:** Salida del programa que implementa el criptoanálisis lineal para el DES de 4 rondas

### 3.2.1. Complejidad y conclusiones

Al igual que el criptoanálisis diferencial, el criptoanálisis lineal es un ataque **probabilístico**, por lo que es interesante preguntarse por el número de parejas de textos planos y cifrados que debemos proporcionar al ataque para poder obtener la clave correcta con una alta probabilidad. Este número estará determinado por el módulo del sesgo de la probabilidad de ocurrencia de la expresión lineal elegida. En [5] Matsui establece que, bajo ciertas hipótesis de normalidad e independencia (principalmente independencia de las salidas de las S-Boxes), la probabilidad de éxito del algoritmo 3.1 viene dada por la siguiente expresión, donde $q^{(i)}$ es la probabilidad de que los bits de la salida de la función $F$ en la expresión lineal usando como clave $K_n^{(i)}$ coincidan con los obtenidos al usar la clave correcta $K_n$, es decir, la probabilidad de que $F(X, K_n^{(i)})$ y $F(X, K_n)$ coincidan en dichos bits, donde $p$ es la probabilidad de la expresión usada y $P$ el número de textos usados:

$$\int_{x=2\sqrt{P}|p-1/2|}^{\infty} \left( \prod_{K_n^{(i)} \neq K_n} \int_{-x-4\sqrt{P}(p-1/2)q^{(i)}}^{x+4\sqrt{P}(p-1/2)(1-q^{(i)})} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy \right) \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \tag{3.6}$$

No es posible calcular esta expresión de manera directa, por lo que se recurre a integrarla de manera numérica, obteniendo las estimaciones de la tabla 3.2, donde $p$ es la probabilidad de ocurrencia de la expresión lineal usada (o equivalentemente, $|p - 1/2|$ es su sesgo en valor absoluto).

| Parejas | Éxito( %) |
|---|---|
| $2\|p-1/2\|^{-2}$ | $48{,}6\,\%$ |
| $4\|p-1/2\|^{-2}$ | $78{,}5\,\%$ |
| $8\|p-1/2\|^{-2}$ | $96{,}7\,\%$ |
| $16\|p-1/2\|^{-2}$ | $99{,}9\,\%$ |

**Tabla 3.2:** Probabilidad de éxito del algoritmo en función del número de parejas de textos

En nuestra implementación hemos fijado el número de parejas de textos planos y cifrados al algoritmo en $8|p - 1/2|^{-2}$, ya que nos ofrece una probabilidad de éxito bastante elevada, además del hecho de que Matsui extablece en [5] que sus resultados experimentales estiman que estas proba-

bilidades son algo superiores a las teóricas, probablemente debido a las hipótesis de normalidad e independencia tomadas en el cálculo teórico realizado. En caso de que el algoritmo falle tendremos que ejecutarlo más veces hasta obtener la clave correcta (que si estamos usando los textos planos y cifrados correspondientes a una probabilidad del 96,7 % no debería fallar muy a menudo).

A modo de comparativa con la tabla 3.1, añadimos en esta sección una tabla 3.3 con las parejas de textos planos y cifrados necesarios para que el algoritmo encuentre la clave correcta con una probabilidad de aproximadamente $96{,}7\,\%$ en función del número de rondas del DES que queramos criptoanalizar.

| **Rondas** | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Parejas** | $2^{7,7}$ | $2^{11,1}$ | $2^{14,4}$ | $2^{19,1}$ | $2^{21,1}$ | $2^{24,4}$ | $2^{29,1}$ | $2^{31,8}$ | $2^{33,1}$ | $2^{36,5}$ | $2^{39,8}$ | $2^{44,5}$ | $2^{46,5}$ |

**Tabla 3.3:** Parejas necesarias para romper el DES en función del número de rondas mediante criptoanálisis lineal con una probabilidad del 96,7 %

Como podemos apreciar (contrastando las tablas 3.1 y 3.3), para un número de rondas inferior a 9 el criptoanálisis diferencial requiere de un menor número de parejas, mientras que a partir de 9 rondas es el criptoanálisis lineal el que necesita menos rondas para poder aplicarse. Es importante notar también que para el caso del DES de 16 rondas este ataque mejora las condiciones frente al ataque de búsqueda por fuerza bruta ($2^{56}$ frente a $2^{46,5}$).

## 3.3. Aprendizaje automático

En esta sección hablaremos de los algoritmos de cifrado y las implementaciones de los mismos que estudiaremos utilizando aprendizaje automático, además de introducir una métrica auxiliar que usaremos en ciertos experimentos en el capítulo 4. Una breve introducción a las librerías auxiliares usadas para implementar estos experimentos se puede encontrar en el anexo C.

### 3.3.1. Pruebas de fortaleza sobre los algoritmos DES, AES y Speck

Para llevar a cabo los experimentos primero necesitamos conjuntos de datos de los que aprender. Para ello necesitamos implementaciones de algoritmos de cifrado. En nuestros experimentos analizaremos tres algoritmos de cifrado: DES, AES y Speck.

Ya hemos hablado del DES en capítulos anteriores y en el anexo A, además de haber implementado en Python una versión propia del mismo, que es la que usaremos para generar los conjuntos de datos relativos a este cifrado.

El AES es el algoritmo de cifrado que sustituyo al DES tras quedar patente que este era rompible por fuerza bruta en apenas un día [20]. En el anexo B se puede encontrar una descripción detallada

de su funcionamiento interno. Fue estandarizado por el NIST en 2001 [21], y es un caso concreto del cifrado Rijndael creado por Joan Daemen y Vincent Rijmen [22]. Soporta claves de 128, 192 y 256 bits, mientras que su tamaño de bloque es constante (128 bits). Según el tamaño de clave escogido, el número de rondas a ejecutar es de, respectivamente, 10, 12 y 14. Para generar conjuntos de datos del AES usaremos la implementación en C de Rijmen y Daemen [1] con ligeras modificaciones del código fuente para poder acceder a datos intermedios del cifrado.

Por último, el Speck es un algoritmo de tipo *lightweight* publicado por la National Security Agency (NSA) en 2013 junto con Simon, estando el primero optimizado para implementaciones en software mientras que el segundo está optimizado para implementaciones en hardware. Las operaciones de ronda, en contraste con las del DES, son muy rápidas (suma, XOR y rotaciones de bits), aunque tiene que llevar a cabo un mayor número de rondas (entre 22 y 34 según el tamaño de bloque y clave escogidos). Soporta tamaños de bloque de 32, 48, 64, 96 y 128 bits y tamaños de clave de 64, 72, 96, 128, 144, 192 y 256 bits, lo que lo hace muy versátil y escalable en función de la seguridad y la potencia de cómputo que se necesite en cada caso. Para generar conjuntos de datos de este cifrado partiremos de la implementación en Python realizada por Calvin McCoy [2] (con ciertas modificaciones que hemos realizado al código para poder acceder a datos intermedios del cifrado), que también ha realizado implementaciones del Speck y del Simon en VHSIC Hardware Description Language (VHDL) y en C.

### 3.3.2. Métrica de evaluación

Prácticamente todas las herramientas que usaremos en los experimentos (sin contar los conjuntos de datos que generaremos nosotros) nos las proporcionará Keras, pero hay un aspecto particular de nuestros conjuntos de entrenamiento que harán necesario la implementación de una nueva métrica.

Las métricas, como su propio nombre indican, son funciones que miden la manera en la que las predicciones de la red neuronal (u otros modelos de aprendizaje) se ajustan a la salida real. Generalmente, si se está trabajando en un problema de regresión se suele usar como métrica el error cuadrático medio (cuanto menor sea mejor será el ajuste) y en caso de que se trate de un problema de clasificación se suele usar el porcentaje de acierto o *accuracy*.

Nuestro problema se enmarca en el conjunto de problemas de clasificación, pero el porcentaje de acierto en bruto sobre las clases nos clasificará de igual manera una salida que coincida en todos los bits menos en uno con la salida esperada y otra que no coincida en ningún bit. Por ello usaremos una métrica similar al porcentaje de acierto, pero no por clase, sino por bit. Es decir, calcularemos el **porcentaje de bits** que la red predice correctamente. La expresión matemática de esta métrica se puede apreciar en la siguiente ecuación, donde $M$ es el tamaño del conjunto de datos a medir, $N$ es

[1] www.embeddedsw.net/Cipher_Reference_Home.html#AES

[2] www.github.com/inmcm/Simon_Speck_Ciphers/blob/master/Python/simonspeckciphers/speck/speck.py

la longitud en bits de la salida del algoritmo, $pred(i,j)$ es el bit $j$ del ejemplo $i$ predicho por la red neuronal y $real(i,j)$ es el bit $j$ del ejemplo $i$ real:

$$Metrica = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} pred(i,j) \oplus real(i,j)}{M * N}. \tag{3.7}$$

Usaremos esta métrica en todos los experimentos cuya salida sea la salida de un algoritmo de cifrado (o, en algunos casos, la salida de alguna operación intermedia del cifrado). En el código esta métrica se implementa en la función `custom_accuracy`.

# 4

# Experimentos y resultados

Comenzaremos este capítulo con un primer experimento en el que comprobaremos de forma empírica las estimaciones teóricas de la probabilidad de éxito del criptoanálisis lineal realizadas en la sección 3.2.1, tras lo cual llevaremos a cabo una serie de experimentos con redes neuronales en **Keras** atendiendo a **diferentes características** de los cifrados (número de rondas, datos intermedios del cifrado, etc.) con el objetivo de servir como una **herramienta para comparar la fortaleza y robustez** de diferentes cifrados o de partes del mismo. Para ello, hemos realizado una serie de pruebas previas de tiempo de ejecución, estableciendo que en el hardware en el que se van a realizar los experimentos (Nvidia GTX GeForce 1060 6GB y AMD Ryzen 5 3600 6-Core) el entrenamiento de las redes con un conjunto de tamaño $1000000 \approx 2^{20}$ (dividiéndolo en $700000$ para el entrenamiento y $300000$ para las pruebas) tarda un tiempo razonable (varias horas) sin llegar a ser inviable, por lo que, **salvo que se indique lo contrario, trabajaremos con conjuntos de datos de ese tamaño**. Cabe destacar que los patrones de cada conjunto de entrenamiento variarán según el algoritmo y el experimento concretos.

El conjunto de datos variará según el experimento, pero en muchos de ellos haremos uso de valores intermedios (entre las rondas) de los cifrados que hemos obtenido modificando el código (simulando un *sniffer*) de los algoritmos de cifrado que vamos a estudiar.

También es importante elegir los modelos de redes neuronales a usar en los experimentos. En todos ellos comenzaremos con un **perceptrón multicapa**, ya que es un modelo sencillo que suele dar buenos resultados. En el caso de que en algún experimento los resultados de este modelo fuesen insatisfactorios, probaremos con otros modelos más complejos. En particular, una configuración que nos ha dado buenos resultados ha sido un **perceptrón multicapa** con **una capa oculta de 2048 neuronas** con función de activación Rectified Linear Unit (ReLu) , y con función de activación **sigmoide** en las neuronas de la capa de salida (la dimensión de esta capa de salida variará según el cifrado que estemos analizando), y es la configuración que se usará salvo que se indique lo contrario.

Por último, queremos destacar que en todos los experimentos se ha establecido la **semilla del generador de números aleatorios en un número fijo** (aunque se ha ejecutado también con otros valores para asegurarnos la validez del experimento), con lo que estos experimentos **pueden ser reproducidos** por cualquier usuario obteniendo los mismos resultados que los aquí expuestos.

## 4.1. Comprobación empírica de la probabilidad de éxito del criptoanálisis lineal

Comenzaremos con un sencillo experimento, en el que comprobaremos de forma empírica la probabilidad de éxito del criptoanálisis lineal para el DES de 4 rondas descrito en la sección 3.2 para diferentes valores del número de textos planos y sus correspondientes textos cifrados (que denominaremos $N$) suministrados al algoritmo. Para ello partiremos de los (aproximadamente) 210 textos planos y cifrados indicados por la tabla 3.3 obtenidos según la estimación teórica de Matsui para tener un éxito del 96,7 % y realizaremos **3000 criptoanálisis lineales** (generando diferentes textos planos y cifrados aleatorios para cada uno de ellos) para valores de $N$ comenzando en 21 e incrementándose de 21 en 21 hasta 210 midiendo el número de ataques que obtienen de manera exitosa la clave de cifrado, obteniendo así una estimación empírica de la probabilidad de éxito de este ataque para esos valores de $N$ en el DES de 4 rondas. El código correspondiente a este experimento se puede encontrar en el fichero `DES_lin_exp.py`.

### 4.1.1. Resultados

Los resultados de este experimento se pueden apreciar en la figura 4.1.

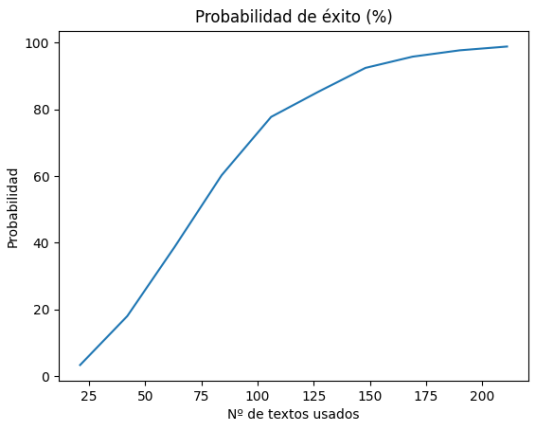



**Figura 4.1:** Resultado del experimento de la comprobación empírica de la probabilidad de éxito del criptoanálisis lineal para el DES de 4 rondas

Como podemos ver, a medida que aumentamos el número de textos planos con sus correspondientes textos cifrados la probabilidad de éxito aumenta de manera significativa. Comenzamos con una probabilidad del 3.37 % con $N = 21$ y alcanzamos una probabilidad del 98,83 % con $N = 210$, 2 puntos porcentuales superior a la estimada teóricamente en la tabla 3.2, lo que coincide con las anotaciones de Matsui en [5], en las que indica que él también detectó esta divergencia entre las probabilidades teórica y empírica del éxito del criptoanálisis lineal (aunque sin aportar datos concretos).

## 4.2. Emulación del comportamiento total del cifrado

Nuestro siguiente experimento consistirá en una primera aproximación a la combinación del mundo de la criptografía con el del aprendizaje automático. Para cada algoritmo de cifrado que vamos a estudiar, generaremos un conjunto de datos de textos planos y cifrados (por tanto estamos en el contexto de un KPA, concepto introducido en la sección 2.1) de una ronda para una **clave fijada**, y entrenaremos una red neuronal con esos datos. Si el porcentaje de acierto es distinto del 50 % (es decir, si nuestra red es capaz de emular parcialmente el funcionamiento del algoritmo de cifrado para 1 ronda), llevaremos a cabo el experimento con 2 rondas, así hasta alcanzar la ronda en la cual no seamos capaces de obtener un porcentaje de acierto distinto del 50 %.

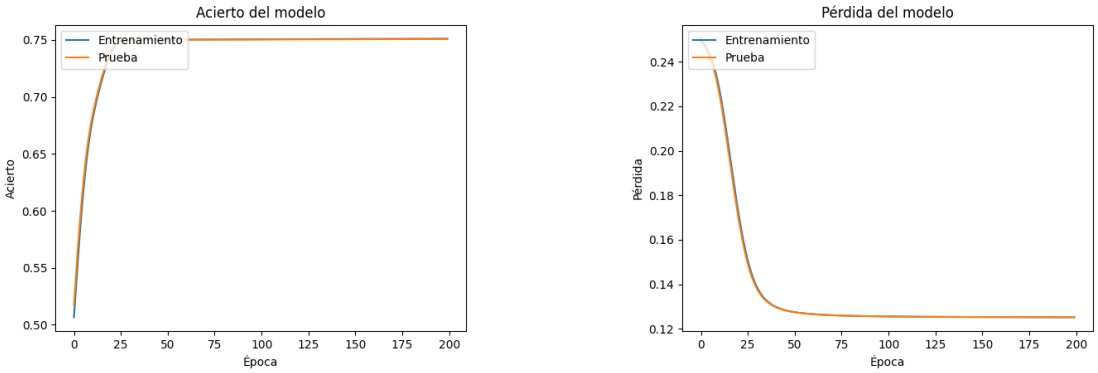
Como mencionamos en la introducción comenzaremos entrenando un perceptrón multicapa. Los principales parámetros a ajustar en este modelo son el número de capas ocultas, el número de neuronas en cada capa, el optimizador y la función de activacion de cada neurona. Ajustaremos estos parámetros hasta obtener unos resultados razonables en el DES, tras lo cual usaremos ese mismo modelo para aprender de los conjuntos de datos de los otros dos algoritmos de cifrado, teniendo así una manera de comparar la robustez y la seguridad relativa de los tres. Además, para certificar el porcentaje de acierto obtenido en cada ejecución llevaremos a cabo una **validación cruzada de 5 iteraciones**. El código fuente correspondiente a la generación de los conjuntos de datos y al experimento, además de las gráficas, los ficheros .csv con los datos de los entrenamientos y las redes neuronales entrenadas se pueden encontrar en el directorio `Directo`.

### 4.2.1. Resultados

Como hemos mencionado, comenzamos el experimento buscando una red que nos proporcione buenos resultados para una ronda del DES. Se han realizado pruebas con varias configuraciones diferentes, principalmente de perceptrones multicapa, pero por brevedad expondremos una muestra con los mejores resultados. En las figuras 4.2 y 4.3 podemos ver los resultados al usar una red de tres capas ocultas (la configuración exacta se indica en el pie de figura) variando el optimizador (en estas figuras la gráfica naranja representa el valor de la variable medida en el conjunto de test de manera meramente ilustrativa).

Como podemos apreciar, el optimizador Adaptative Moment Estimation (ADAM) [23], pese a generar una gráfica algo más irregular, permite que el aprendizaje supere la barrera del 75 % de porcentaje de acierto en la que se queda la versión con el optimizador Stochastic Gradient Descent (SGD) .

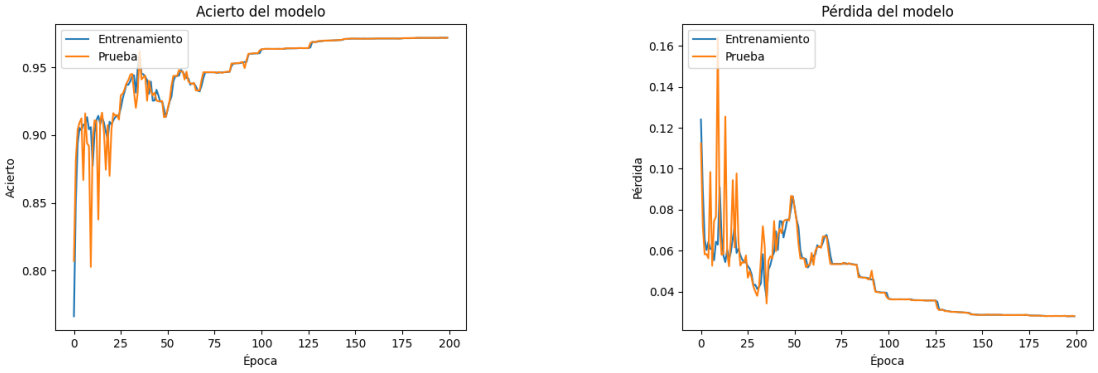
Una vez visto que el optimizador ADAM es el que mejores resultados nos proporciona probamos a modificar el número de capas ocultas y el número de neuronas de cada capa oculta, obteniendo los mejores resultados en una configuración de una única capa oculta con 2048 neuronas. Ahora fijamos esta arquitectura de red y la entrenamos con diferentes tamaños de rondas (a la red neuronal se le




(a) Porcentaje de acierto

(b) Pérdida

**Figura 4.2:** Gráficas de porcentaje de acierto y error para el conjunto de datos del DES de una ronda usando el optimizador SGD una configuración de tres capas ocultas con 256, 512 y 1024 neuronas respectivamente, 200 épocas y 2000 de tamaño de lote




(a) Porcentaje de acierto

(b) Pérdida

**Figura 4.3:** Gráficas de porcentaje de acierto y error para el conjunto de datos del DES de una ronda usando el optimizador ADAM una configuración de tres capas ocultas con 256, 512 y 1024 neuronas respectivamente, 200 épocas y 2000 de tamaño de lote

introduce el texto plano y devuelve su predicción del texto cifrado para la versión del algoritmo de cifrado de ese número de rondas) del DES, del AES y del Speck usando validación cruzada de 5 iteraciones. Los resultados obtenidos se pueden ver en la tabla 4.1 (para el Speck hemos comparado las versiones correspondientes a un tamaño de bloque de 64 y de 96 bits manteniendo un tamaño de la clave constante de 96 bits).

| Cifrado | Porcentaje de acierto |
|---|---|
| DES 1R | 99.90 % (+/- 0.02 %) |
| DES 2R | 76.25 % (+/- 0.07 %) |
| DES 3R | 50.04 % (+/- 0.01 %) |
| AES 1R | 50.00 % (+/- 0.00 %) |
| Speck 64 1R | 99.72 % (+/- 0.01 %) |
| Speck 64 2R | 49.99 % (+/- 0.02 %) |
| Speck 96 1R | 99.46 % (+/- 0.04 %) |
| Speck 96 2R | 49.99 % (+/- 0.01 %) |

**Tabla 4.1:** Porcentaje de acierto y varianza (entre paréntesis) del experimento 1 en diferentes cifrados usando validación cruzada de 5 iteraciones

A la vista de la tabla, podemos concluir que la función de ronda del DES es la más "predecible" frente a este análisis, ya que tarda 3 rondas hasta bajar al 50 % de porcentaje de acierto de esta red neuronal, mientras que la del AES es muy resistente, cayendo al 50 % en tan solo una ronda. Sorprende también el hecho de que la función de ronda del Speck caiga al 50 % en tan solo 2 rondas mientras que el DES tarda 3, lo que parece indicar que el Speck es un algoritmo que, pese a ser de tipo *lightweight*, es bastante robusto. Por último, vemos que entre las dos versiones de Speck analizadas apenas hay diferencias, lo que nos indica que este ataque no es afectado en gran medida por la diferencia de los tamaños de bloque del cifrado.

## 4.3. Emulación ronda a ronda del cifrado

Vistos los resultados del primer experimento, podemos apreciar que ese primer enfoque, aunque útil, apenas es capaz de avanzar de la segunda ronda de ninguno de los algoritmos. Por ello, vamos a tomar un camino similar pero que en un principio debería ser más sencillo para las redes neuronales. En este caso entrenaremos varias redes neuronales, una por cada ronda del algoritmo, y estudiaremos hasta que punto este conjunto de redes neuronales enlazadas puede simular el algoritmo de cifrado para el número total de rondas. Para obtener los datos intermedios hemos modificado los códigos de los tres cifrados accediendo a los bits de los valores a cada ronda de los algoritmos de cifrado. De igual manera que en el experimento anterior, los textos cifrados han sido obtenidos usando la misma clave de cifrado.

El conjunto de datos a usar en este experimento aumenta bastante, ya que necesitamos almacenar no solo los bits del bloque inicial y final, sino los bits de todos los valores intermedios. Mantendremos el mismo tamaño de los conjuntos de entrenamiento y pruebas (700000 y 300000, respectivamente), aunque hemos decidido prescindir de la validación cruzada ya que el aumento de tiempo sería inasumible y, como hemos visto en el primer experimento, este tipo de datos no sufre apenas de sobreajuste y apenas pierde porcentaje de acierto en el conjunto de pruebas frente al obtenido en el conjunto de entrenamiento. El código fuente correspondiente a la generación de los conjuntos de datos y al experimento, además de las gráficas, los ficheros .csv con los datos de los entrenamientos y las redes neuronales entrenadas se pueden encontrare en el directorio `Iterativo`.

### 4.3.1. Resultados

Comenzamos simulando las 16 rondas del DES entrenando 16 redes neuronales con la estructura que heredamos del experimento 1 (perceptrón multicapa con una capa oculta de 2048 neuronas y optimizador ADAM). Cada red $i$ es entrenada con los valores de $C_i$ como entrada y $C_{i+1}$ como salida, siendo $C_0$ el texto plano y $C_i$ los bits tras haber sido ejecutada la ronda $i$. Tras ello, se comprueba el porcentaje de acierto que tiene el conjunto de las $N$ primeras redes combinadas para $N$ desde 1 hasta 16, obteniendo los resultados de la tabla 4.2.

| Rondas | Acierto | Rondas | Acierto | Rondas | Acierto | Rondas | Acierto |
|---|---|---|---|---|---|---|---|
| 1 | 99.85 % | 5 | 99.23 % | 9 | 98.79 % | 13 | 99.31 % |
| 2 | 99.03 % | 6 | 99.42 % | 10 | 99.15 % | 14 | 98.75 % |
| 3 | 98.88 % | 7 | 99.16 % | 11 | 98.84 % | 15 | 98.90 % |
| 4 | 98.98 % | 8 | 99.16 % | 12 | 99.22 % | 16 | 99.06 % |

**Tabla 4.2:** Porcentaje de acierto del experimento 2 en las 16 rondas del DES

Vemos que, al ser capaz nuestro modelo de red neuronal de aprender una ronda del DES con facilidad, el conjunto de las 16 redes neuronales enlazadas predice correctamente el 99.06 % de los bits de la salida del DES completo. Esto no quiere decir, ni por asomo, que hemos roto el algoritmo de cifrado con estas 16 redes neuronales, ya que estamos usando una gran cantidad de valores intermedios del algoritmo de cifrado que deberían extraerse de la memoria del ordenador donde se estén cifrando los textos planos mientras se esté produciendo este cifrado. Además esta red se ha entrenado con textos planos y cifrados provenientes de un DES con la misma clave, por lo que un inocente cambio de clave dejaría inservibles estas redes neuronales.

Sin embargo, aunque a nivel práctico este experimento no sea muy útil, sí nos da una idea de cómo la información del criptograma se va difundiendo y confundiendo con las subclaves [24].

Procedemos ahora a realizar el mismo experimento con el AES, aunque los resultados obtenidos no consiguen alejarse del 50 %, ya que como hemos visto en el experimento anterior nuestra red no

es capaz ni tan siquiera de aprender una ronda del AES, lo que deja patente que este algoritmo de cifrado es muy robusto.

A continuación realizamos el experimento sobre el Speck de 64 y 96 bits de tamaño de bloque con 96 bits de tamaño de clave. Los resultados obtenidos se exponen en la tabla 4.3, donde se exponen solo las rondas pares a fin de economizar espacio, además de no ser los resultados de las rondas impares muy diferentes a los de las rondas pares.

| Rondas | 64/96 | 96/96 | Rondas | 64/96 | 96/96 | Rondas | 64/96 | 96/96 |
|---|---|---|---|---|---|---|---|---|
| 2 | 97.69 % | 96.28 % | 12 | 97.62 % | 96.25 % | 22 | 97.67 % | 96.23 % |
| 4 | 97.66 % | 96.54 % | 14 | 97.78 % | 95.80 % | 24 | 97.68 % | 96.32 % |
| 6 | 97.73 % | 96.55 % | 16 | 97.64 % | 95.92 % | 26 | 97.68 % | 96.54 % |
| 8 | 97.72 % | 95.98 % | 18 | 97.63 % | 96.23 % | 28 | - % | 96.03 % |
| 10 | 97.69 % | 96.27 % | 20 | 97.69 % | 95.94 % | | | |

**Tabla 4.3:** Porcentaje de acierto del experimento 2 en Speck 64/96 y Speck 96/96

Como se puede apreciar, tenemos unos resultados ligeramente peores que los del DES, aunque siguen siendo buenos, ya que nuestro conjunto de 26 redes neuronales consigue imitar el funcionamiento del Speck de 64 bits de tamaño de bloque con un porcentaje de acierto del 97,68 % y nuestro conjunto de 28 redes neuronales consigue imitar el funcionamiento del Speck de 96 bits de tamaño de bloque con un porcentaje de acierto del 96.03 %. Es decir, para un tamaño de clave constante (96 bits), aumentando el tamaño de bloque y llevando a cabo 2 rondas más del algoritmo de cifrado, el algoritmo ha aumentado en robustez y esta ha sido detectada por las redes neuronales.

Ya que la operación que suele ser más “interesante” a la hora de atacar un algoritmo de cifrado es la función de descifrado, hemos entrenado 16 redes neuronales con el conjunto de datos del DES de manera inversa, obteniendo resultados prácticamente idénticos a los del DES al entrenarlo con los datos en el orden directo.

## 4.4. Detección de diferenciales en el DES

En este cuarto experimento nos centraremos en el algoritmo de cifrado DES y en el criptoanálisis diferencial estudiado en los capítulos 2 y 3. El experimento consistirá en el aprendizaje de diferenciales de salida del algoritmo que provengan de un diferencial de entrada concreto, es decir, se le suministrarán diferencias en bits de parejas de textos cifrados, la mitad de ellas provenientes de parejas de textos planos cuya diferencia en bits sea un diferencial escogido por nosotros (usaremos los vistos en la sección 3.1) y la otra mitad provenientes de textos planos aleatorios. Por tanto, estamos construyendo un **distinguidor diferencial**. De esta manera, si conseguimos un porcentaje de acierto significativamente alto podremos definir una mejora del criptoanálisis diferencial, pasando de ser un CPA a un COA, es

decir, que podremos atacar el algoritmo de cifrado tan solo conociendo un conjunto de textos cifrados, **sin conocer el texto plano del que provienen**.

Este experimento cambia el enfoque con respecto a los anteriores, ya que en este caso no trataremos de averiguar los bits del texto cifrado, sino que dada una diferencia entre dos textos cifrados se tratará de detectar si la diferencia de los textos planos correspondientes a cada uno de ellos es una determinada. Por tanto, ahora tenemos entre manos un problema habitual de clasificación, donde la salida será 1 si el diferencial de salida proviene del diferencial de entrada escogido y 0 en caso contrario. Por ello, en este caso hemos usado como métrica el porcentaje de acierto habitual, y como función de pérdida la conocida como *Binary Cross-entropy*.

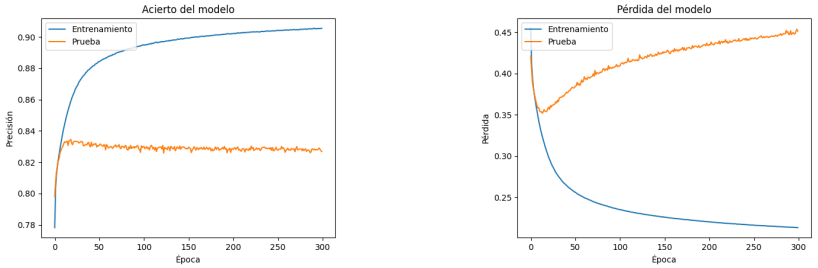
El código fuente correspondiente a la generación de los conjuntos de datos y al experimento, además de las gráficas, los ficheros .csv con los datos de los entrenamientos y las redes neuronales entrenadas se pueden encontrare en el directorio `Diferencial`.

### 4.4.1. Resultados

Comenzamos con la generación de un conjunto de datos de un DES de 4 rondas en el que la mitad de los ejemplos provienen de parejas de textos cuya diferencia es la usada en la sección 3.1.2 ($2000000000000000_x$) y la otra mitad proviene de parejas de textos planos cuya diferencia es aleatoria. El resultado que nos da nuestra configuración habitual de un perceptrón multicapa de una capa oculta con 2048 neuronas y optimizador ADAM nos da los resultados de la figura 4.4. Además, hemos incluido una métrica más que nos permitirá evaluar el porcentaje de diferenciales provenientes del diferencial escogido que ha detectado la red frente al total de ejemplos del diferencial escogido (*recall* o exhaustividad), es decir, la tasa de detección del diferencial.

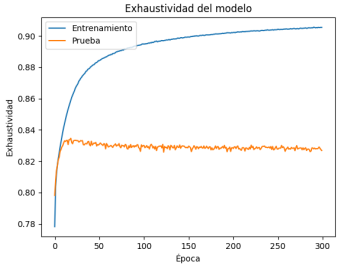
Como podemos ver el porcentaje de acierto y la exhaustividad son bastante altas, alcanzando ambas el 82,81 %, por lo que nuestra sencilla red es capaz de detectar con cierta precisión diferenciales procedentes del diferencial escogido, lo que nos permitirá llevar a cabo la modificación del ataque que veremos en el siguiente párrafo. Es importante notar que hay cierto sobreajuste que apreciamos en la figura 4.4(a). Para reducirlo hemos tratado de aplicar regularización L1 y *dropout* [25], pero ninguna de estas herramientas nos ha proporcionado mejora alguna.

Veamos ahora como podemos transformar el criptoanálisis diferencial en un COA. Tan solo tenemos que combinar ambos elementos: el criptoanálisis diferencial y la red neuronal entrenada. Se partirá de un conjunto de textos cifrados cuyas diferencias por parejas se suministrarán a la red, formando un nuevo conjunto con las parejas de textos planos que la red clasifica como provenientes del diferencial objetivo. Ahora, si el conjunto inicial tenía suficientes parejas de textos cifrados, en el segundo conjunto tendremos cierto número de parejas de textos cifrados que provienen del diferencial objetivo (concretaremos las cantidades *suficientes* y *cierto número* más adelante). Tras ello, tan solo tenemos




(a) Porcentaje de acierto

(b) Pérdida




(c) Exhaustividad

**Figura 4.4:** Gráficas de porcentaje de acierto, pérdida, precisión y exhaustividad para el conjunto de datos del DES de cuatro rondas usando el optimizador ADAM una configuración de una capa oculta con 2048 neuronas, 300 épocas y 1000 de tamaño de lote

que llevar a cabo el criptoanálisis diferencial varias veces suministrando diferentes subconjuntos de tamaño correspondiente al indicado en la tabla 3.1 según el número de rondas de este conjunto, hasta que uno de estos subconjuntos haga que el criptoanálisis diferencial nos devuelva la clave.

Ya que la probabilidad de que una pareja de textos planos escogidos aleatoriamente cumpla que su diferencial es el diferencial escogido es $1/2^{64}$, y necesitamos que haya como mínimo tantas parejas en el segundo conjunto como las necesarias para que el criptoanálisis diferencial pueda obtener la clave, tenemos que el número de parejas de textos planos necesarios es $m \cdot p \cdot 2^{64}$ siendo $m$ el número de parejas de textos cifrados necesarios para que el criptoanálisis diferencial tenga éxito y $p$ la exhaustividad de la red neuronal para el diferencial escogido. Esto se traduce en que necesitamos $n$ textos planos siendo $n$ el menor número tal que se cumple que $T_n$ es mayor que $m \cdot p \cdot 2^{64}$ donde $T_n$ es la sucesión de los números triangulares (que nos dan el número de parejas que se pueden hacer con $n$ elementos).

A fin de comparar ambos casos del criptoanálisis diferencial estudiados, se ha entrenado otra red neuronal (con las mismas especificaciones que la anterior) usando diferenciales de salida del DES de 6 rondas que provienen del diferencial $4008000004000000_x$, obteniendo un porcentaje de acierto y una exhaustividad de un 52,34 % aproximadamente.

Como conclusión, este ataque es completamente impracticable ya que necesita una cantidad des-

comunal de textos cifrados, pero sí es interesante a nivel teórico, estableciendo que se podría romper este algoritmo de cifrado usando únicamente textos cifrados. Además, cuanto más fácil sea detectar diferenciales, menos robusto será el algoritmo, por lo que es otra herramienta que nos permite evaluar la fortaleza de diferentes algoritmos.

## 4.5. Evaluación de las funciones de la ronda del AES

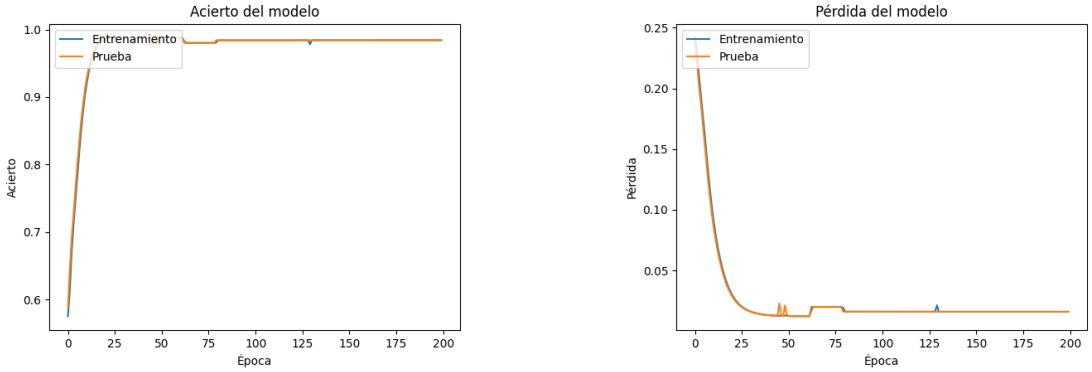
Para el quinto experimento vamos a centrarnos en el algoritmo de cifrado AES, ya que es el que más robustez ha presentado para en los primeros experimentos. Enfocaremos este experimento hacia el estudio de la robustez que confieren al algoritmo de cifrado sus diferentes funciones internas, y este enfoque se inspira en lo observado en los resultados del segundo experimento 4.3.1, donde vimos que la red neuronal aprendía con un alto porcentaje de acierto el comportamiento de la última ronda del AES, mientras que tenía unos resultados significativamente peores en el resto de rondas.

Cada ronda del AES consiste en la aplicación de cuatro funciones: *SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*. La última ronda del AES tiene un comportamiento ligeramente diferente al resto de rondas, omite la función *MixColumns*. El experimento consistirá en eliminar la función *MixColumn* y la función *ShiftRows* y evaluar cómo influyen esas modificación en el aprendizaje por parte de una red neuronal del comportamiento una única ronda del cifrado completo. No consideramos en este experimento la omisión de la función *SubBytes*, ya que es la que aporta la no linealidad al cifrado y sin ella el cifrado sería rompible trivialmente con unas cuantas parejas de texto plano y cifrado, ni tampoco la omisión de la función *AddRoundKey*, ya que en ese caso cada texto plano se cifraría siempre de la misma forma y el algoritmo perdería la aparente aleatoriedad que busca conseguir con los diferentes valores de la clave. El código fuente correspondiente a la generación de los conjuntos de datos y al experimento, además de las gráficas, los ficheros .csv con los datos de los entrenamientos y las redes neuronales entrenadas se pueden encontrare en el directorio `ModificacionesRonda`.

### 4.5.1. Resultados

Primero generamos los conjuntos de datos de una ronda omitiendo la función *MixColumns* y otro omitiendo la función *ShiftRows*, y tras ello entrenaremos la configuración habitual de un perceptrón multicapa con una capa oculta de 2048 neuronas y ADAM como optimizador. Los resultados son los que se pueden ver en las figuras 4.5 y 4.6.

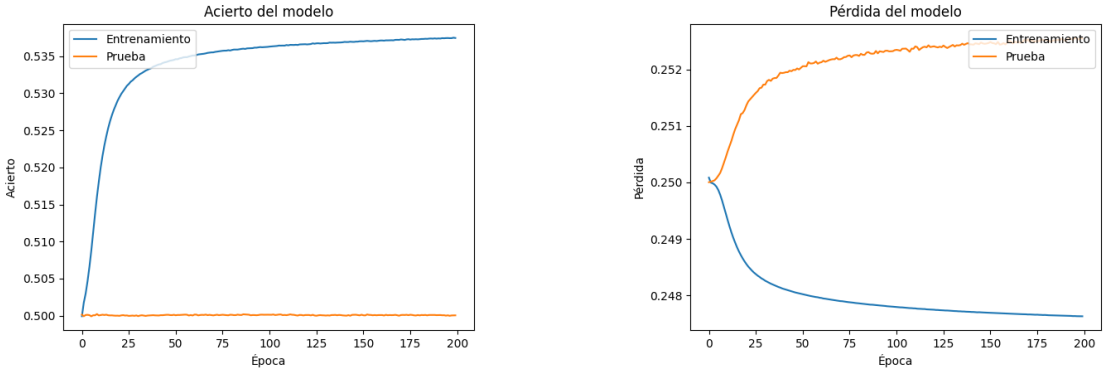
Como podemos ver, al omitir la función *MixColumns* nuestra red neuronal es capaz de aprender con un alto porcentaje de acierto el comportamiento de una ronda, mientras que si omitimos la función *ShiftRows* la red neuronal no consigue una precisión superior al 50 %. Esto podría indicarnos que la función *ShiftRows* apenas aporta robustez al cifrado, pero debemos recordar que los algoritmos de




(a) Porcentaje de acierto

(b) Pérdida

**Figura 4.5:** Gráficas de porcentaje de acierto y error para el conjunto de datos del AES de una ronda sin la función *MixColumns* usando el optimizador ADAM una configuración de una capa ocultas con 2048 neuronas, 200 épocas y 2000 de tamaño de lote

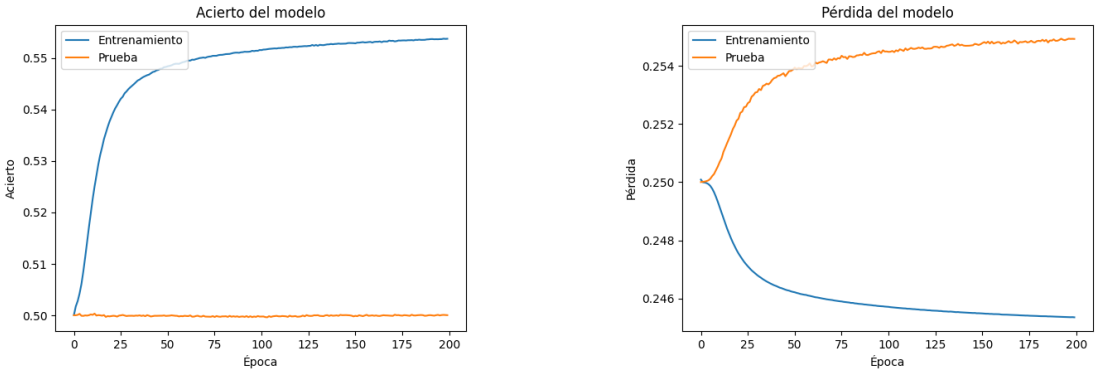



(a) Porcentaje de acierto

(b) Pérdida

**Figura 4.6:** Gráficas de porcentaje de acierto y error para el conjunto de datos del AES de una ronda sin la función *ShiftRows* usando el optimizador ADAM una configuración de una capa ocultas con 2048 neuronas, 200 épocas y 2000 de tamaño de lote

cifrado deben proporcionar dos propiedades, confusión y difusión. Veamos que ocurre si llevamos a cabo más rondas omitiendo la función *MixColumns*. Podemos ver los resultados para el AES de 2 rondas sin *MixColumns* en la figura 4.7.




(a) Porcentaje de acierto

(b) Pérdida

**Figura 4.7:** Gráficas de porcentaje de acierto y error para el conjunto de datos del AES de dos rondas sin la función *MixColumns* usando el optimizador ADAM una configuración de una capa ocultas con 2048 neuronas, 200 épocas y 2000 de tamaño de lote

Vemos que en 2 rondas nuestra red neuronal vuelve a estancarse en el 50 %. Para validar nuestras predicciones hemos repetido estos experimentos usando validación cruzada de 5 iteraciones, cuyos resultados al finalizar el entrenamiento podemos ver en la tabla 4.4.

| Cifrado | Porcentaje de acierto |
|---|---|
| AES 1R sin MC | 98.75 % (+/- 0.07 %) |
| AES 2R sin MC | 50.01 % (+/- 0.01 %) |
| AES 1R sin SR | 49.99 % (+/- 0.01 %) |

**Tabla 4.4:** Porcentaje de acierto y varianza (entre paréntesis) del experimento 4 en diferentes cifrados usando validación cruzada de 5 iteraciones

La principal conclusión que extraemos de este experimento es que la red neuronal nos ha permitido detectar las propiedades que está aportando cada una de las dos funciones, siendo la función *MixColumns* la que aporta confusión y la función *ShiftRows* la que aporta difusión (sin olvidar que las funciones *AddRoundKey* y *SubBytes* aportan difusión). Por tanto, las redes neuronales nos permiten extraer información de las funciones internas de un algoritmo de cifrado sin necesidad de conocer sus fundamentos matemáticos, y que la combinación de **confusión y difusión** [24] es la que aporta robustez y fortaleza al sistema global.

# 5

# Conclusiones y trabajo futuro

## 5.1. Discusión y conclusiones

Tras la realización de este trabajo hemos conseguido cumplir los objetivos planteados al inicio del mismo. En los capítulos 2 y 3 hemos estudiado e implementado las versiones clásicas de los criptoanálisis lineal y diferencial sobre el DES, asimilando sus fundamentos para poder aplicarlos a otros algoritmos de cifrado. Hemos comprobado además que, en el caso concreto del DES, el criptoanálisis lineal es más eficiente que el diferencial para un número alto de rondas, y hemos obtenido (en el primer experimento del capítulo 4) de manera empírica la probabilidad de éxito del criptoanálisis lineal, siendo esta superior incluso a la teórica estimada inicialmente por Matsui. Esto constituye el primer objetivo que nos planteamos al inicio del trabajo.

También nos hemos familiarizado con el *framework* Keras/Tensorflow, y hemos realizado una serie de experimentos en el capítulo 4 abordando diferentes enfoques sobre la unión entre el aprendizaje automático y el criptoanálisis, lo que constituye el segundo objetivo del trabajo. Además, la realización de estos experimentos nos ha permitido observar que, pese a que en una primera instacia pudiera parecer que el aprendizaje automático no se puede aplicar al criptoanálisis (ya que el primero trata de encontrar patrones y el segundo trata de generar la máxima aleatoriedad manteniendo su determinismo), hay varios aspectos de los criptosistemas que podemos aprender en mayor o menor medida mediante el uso de redes neuronales. Esto nos aporta varios mecanismos para comparar la fortaleza de diferentes algoritmos de cifrado atendiendo a diferentes propiedades de los cifrados, además de poder mejorar ataques ya conocidos, como el criptoanálisis diferencial.

Por tanto, podemos concluir que, sin duda alguna, el aprendizaje automático es una herramienta muy importante a tener en cuenta a la hora de evaluar la fortaleza y robustez de nuevos algoritmos de cifrado, ya que nos proporciona tanto evaluaciones de caja negra que se pueden aplicar cualquier algoritmo sin apenas modificaciones específicas como de caja blanca que nos permiten comparar diferentes algoritmos atendiendo a sus especificaciones internas.

Este trabajo puede ser considerado como una introducción al amplio abanico de posibilidades que nos ofrece la combinación de aprendizaje automático con el criptoanálisis. Una de las posibles líneas

de investigación para expandir este proyecto podría ser el diseño de una batería de modelos de redes neuronales que sirva para evaluar de manera unificada y estandarizada la robustez de diferentes algoritmos, ya que en este trabajo se ha experimentado con un número limitado de modelos y configuraciones por las restricciones de hardware y tiempo. Otra posibilidad sería el intentar aplicar el aprendizaje automático a ataques tradicionales de otros algoritmos de cifrado (como podrían ser los criptoanálisis lineal y diferencial del AES) de manera similar a lo realizado con el DES en la sección 4.4 o al trabajo de Gohr [8] en el Speck.

# Bibliografía


[1] H. Feistel, W. A. Notz, and J. L. Smith, "Some cryptographic techniques for machine-to-machine data communications," *IEEE Proceedings*, vol. 63, no. 11, pp. 1545–1554, 1975.

[2] Z. Chen and S. Tavares, "Towards provable security of substitution permutation encryption networks," *Selected Areas in Cryptography, SAC'98*, vol. 1556, 1999.

[3] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, vol. 4, no. 1, pp. 3–72, 1991.

[4] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*. Springer Verlag, first ed., 1993.

[5] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Lecture Notes in Computer Science*, vol. 765, pp. 386–397, 1993.

[6] B. Seunggeun and K. Kwangjo, "Recent Advances of Neural Attacks against Block Ciphers," *Symposium on Cryptography and Information Security*, 2019.

[7] X. Hu and Y. Zhao, "Research on Plaintext Restoration of AES Based on Neural Network," *Security and Communication Networks*, 2018.

[8] A. Gohr, "Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning," *Cryptology ePrint Archive, Report 2019/037*, 2019. https://eprint.iacr.org/2019/037.

[9] A. Benamira, D. Gerault, T. Peyrin, and Q. Q. Tan, "A Deeper Look at Machine Learning-Based Cryptanalysis," *Cryptology ePring Archive, Report 2021/287*, 2021. https://eprint.iacr.org/2021/287.

[10] J. So, "Deep Learning-Based Cryptanalysis of Lightweight Block Ciphers," *Security and Communication Networks*, 2020.

[11] M. Wang, "Differential Cryptanalysis of PRESENT," *Cryptology ePrint Archive, Report 2007/408*, 2007. https://eprint.iacr.org/2007/408.

[12] M. Tunstall, "Practical Complexity Differential Cryptanalysis and Fault Analysis of AES," *Cryptology ePrint Archive, Report 2011/453*, 2011. https://eprint.iacr.org/2011/453.

[13] E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES," *Proceedings of CRYPTO '92*, vol. 740, 1991.

[14] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," *IBM Journal of Research and Development*, vol. 38, no. 3, pp. 243–250, 1994.

[15] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Advances in Cryptology - CRYPTO '90*, pp. 2–21, 1990.

[16] K. Sakamura, W. X. Dong, and H. Ishikawa, "A Study on the Linear Cryptanalysis of AES Cipher," *Journal of the Faculty of Environmental Science and Technology, Okayama University*, vol. 9, no. 1, pp. 19–26, 2004.

[17] D. Shi, L. Hu, S. Sun, and L. Song, "Improved Linear (hull) Cryptanalysis of Round-reduced Versions of KATAN," *Cryptology ePrint Archive, Report 2015/964*, 2015. https://eprint.iacr.org/2015/964.

[18] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc., second ed., 1995.

[19] A. Baksi, J. Reier, Y. Chen, and X. Dong, "Machine Learning Assisted Differential Distinguishers For Lightweight Ciphers," *Design, Automation and Test in Europe Conference (DATE)*, 2021.

[20] E. F. Foundation, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, first ed., 1998.

[21] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Roback, and J. F. Dray, "Advanced Encryption Standard (AES)," *Federal Inf. Process. Stds.*, no. 197, 2001.

[22] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *AES Algorithm Submission*, 1999.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ICLR*, 2015.

[24] C. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[25] S. Haykin, *Neural Networks and Learning Machines*. Prentice Hall, third ed., 2009.

# Acrónimos

**ADAM** Adaptative Moment Estimation.

**AES** Advanced Encryption Standard.

**CCA** Chosen Ciphertext Attack.

**COA** Ciphertext Only Attack.

**CPA** Chosen Plaintext Attack.

**DES** Data Encryption Standard.

**GPU** Graphics Processing Unit.

**IBM** International Business Machines.

**KPA** Known Plaintext Attack.

**LSTM** Long Short-Term Memory.

**NIST** National Institute of Standards and Technology.

**NSA** National Security Agency.

**ReLu** Rectified Linear Unit.

**RNN** Recurrent Neural Network.

**S/N** Signal-Noise Ratio.

**SGD** Stochastic Gradient Descent.

**SPN** Substitution-Permutation Networks.

**VHDL** VHSIC Hardware Description Language.

**XOR** Exclusive Or.

# Apéndices

# A

# Funcionamiento del DES

En este anexo expondremos el funcionamiento interno del DES. Está basado en la estructura de una red de Feistel, es decir, su funcionamiento se basa en la división del texto plano en dos mitades de igual tamaño $(L, R)$ a las que se le aplica repetidas veces la operación indicada por la ecuación A.1, que se conoce como **ronda**. En cada ronda se usa una subclave $K_i$ diferente, todas ellas obtenidas aplicando un algoritmo (diferente para cada implementación) conocido como ***key schedule***. En la figura A.1 se puede ver el la estructura de una red de Feistel general con su operación de cifrado y de descifrado (es importante notar que ambas operaciones son idénticas cambiando exclusivamente el orden de las subclaves).

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus F(R_{i-1}, K_i) \end{aligned} \tag{A.1}$$

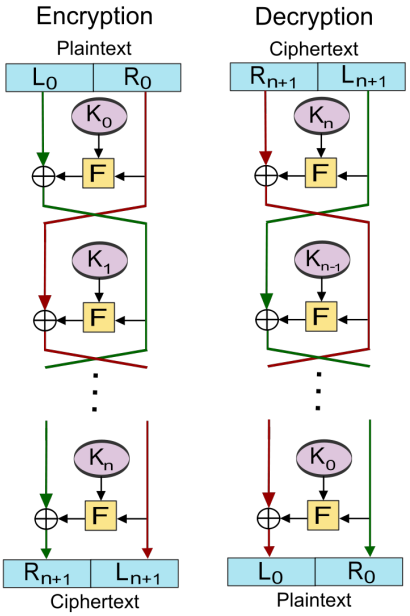
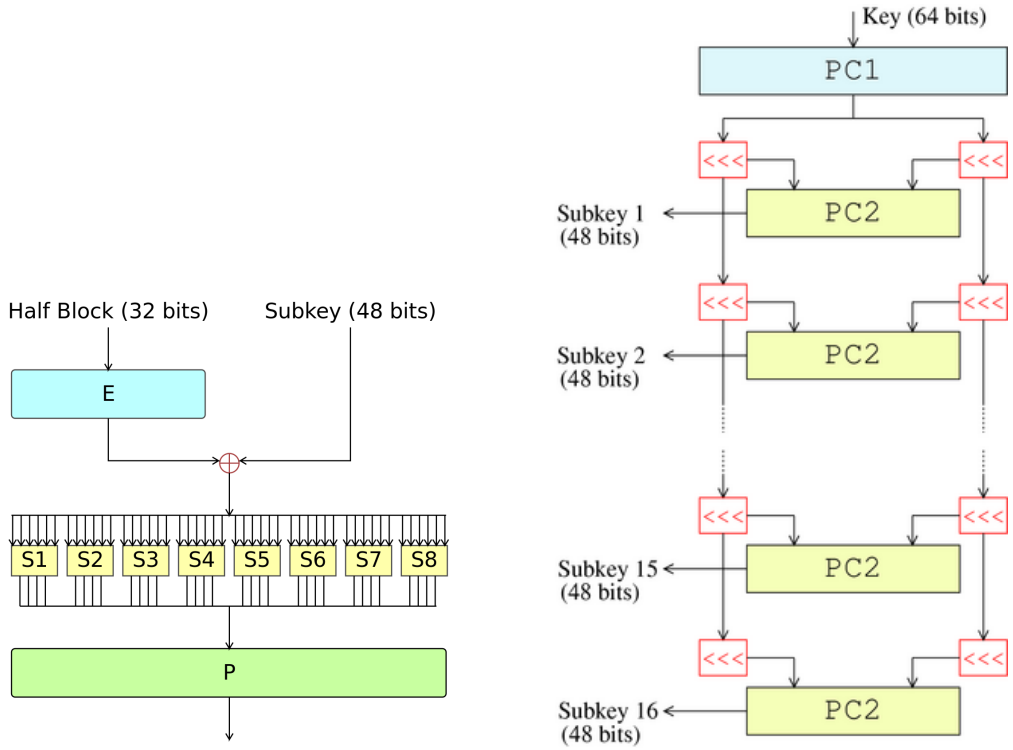



**Figura A.1:** Cifrado y descifrado en una red de Feistel [1]

En el caso concreto del DES el número de rondas a aplicar es 16, la función $F$ es la que se puede ver en la figura A.2(a), y el ***key schedule*** es el que se puede ver en la figura A.2(b).




(a) Función $F$ del DES

(b) *Key schedule* del DES

**Figura A.2:** Función $F$ y ***key schedule*** del DES [2]

En todas las tablas (salvo en la de la S-Box A.2, en el que los números representan directamente la salida de la S-Box) usadas para ilustrar el comportamiento de los componentes del DES cada número representa el **bit en dicha posición de la entrada** de ese componente **siendo 0 el bit menos significativo**.

Los componentes de la función $F$ son:

- Expansión $E$. Genera 48 bits partiendo de los 32 bits de la mitad $L_i$ de la ronda correspondiente. Su funcionamiento es el indicado en la tabla A.1, y nos indica, por ejemplo, que el segundo bit de la salida de la expansión es el bit en la posición 14 de la entrada.
- Cajas de sustitución o S-Boxes $S_i$. Son 8 componentes que sustituyen 6 bits de entrada por 4 bits de salida. Aportan no linealidad y confusión al cifrado. En la tabla A.2 se puede ver el funcionamiento de la S-Box 1, siendo el resto de S-Boxes idénticas en funcionamiento cambiando los números concretos. Por ejemplo, si se introduce como entrada a la S-Box 1 el número $010101_b$, como el primer y el último bit son $01_b$ (segunda fila) y los cuatro bits intermedios son $1010_b$ (undécima columna) la salida será 12, o equivalentemente, $001100_b$.
- Permutación $P$. Reordena los 32 bits obtenidos de las S-Boxes, lo que aporta difusión al cifrado. Se puede ver su funcionamiento en la tabla A.3, donde se nos indica que, por ejemplo, el cuarto bit de la salida de $P$ será el

[1] Extraido de https://en.wikipedia.org/wiki/Feistel_cipher

[2] Extraido de https://en.wikipedia.org/wiki/Data_Encryption_Standard

que se encuentra en la posición 11 de la entrada.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 14 | 22 | 30 | 38 | 46 | 54 | 62 |
| 4 | 12 | 20 | 28 | 36 | 44 | 52 | 60 |
| 2 | 10 | 18 | 26 | 34 | 42 | 50 | 58 |
| 0 | 8 | 16 | 24 | 32 | 40 | 48 | 56 |
| 7 | 15 | 23 | 31 | 39 | 47 | 55 | 63 |
| 5 | 13 | 21 | 29 | 37 | 45 | 53 | 61 |
| 3 | 11 | 19 | 27 | 35 | 43 | 51 | 59 |
| 1 | 9 | 17 | 25 | 33 | 41 | 49 | 57 |

**Tabla A.1:** Expansión $E$ del DES

| **S1** | x0000x | x0001x | x0010x | x0011x | x0100x | x0101x | x0110x | x0111x |
|---|---|---|---|---|---|---|---|---|
| 0yyyy0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 |
| 0yyyy1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 |
| 1yyyy0 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 |
| 1yyyy1 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 |

(a) S-Box $S_1$ del DES (Mitad 1)

| **S1** | x1000x | x1001x | x1010x | x1011x | x1100x | x1101x | x1110x | x1111x |
|---|---|---|---|---|---|---|---|---|
| 0yyyy0 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0yyyy1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 1yyyy0 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 1yyyy1 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

(b) S-Box $S_1$ del DES (Mitad 2)

**Tabla A.2:** Funcionamiento de la S-Box 1 del DES

Los componentes del *key schedule* son:

- Permutación/Elección 1 (PC1). Descarta los bits de paridad y permuta los 56 bits efectivos. Su funcionamiento es el indicado en la tabla A.4, donde, por ejemplo, se nos dice que el bit
- Rotación de bits hacia la izquierda ($<<<$). Rotación de 1 ó 2 (depende de la ronda, el valor concreto para cada una de ellas se puede observar en la tabla A.6) bits hacia la izquierda de cada una de las dos mitades de 28 bits.
- Permutación/Elección 2 (PC2). Descarta 8 bits de los 56 recibidos de las dos rotaciones hacia la izquierda y permuta el resto, generando 48 bits que serán una de las subclaves usadas en el DES. Se puede ver su funcionamiento en la tabla A.5.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 25 | 12 | 11 | 3 | 20 | 4 | 15 |
| 31 | 17 | 9 | 6 | 27 | 14 | 1 | 2 |
| 30 | 24 | 8 | 18 | 0 | 5 | 29 | 23 |
| 13 | 19 | 2 | 26 | 10 | 21 | 28 | 7 |

**Tabla A.3:** Permutación $P$ del DES

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 15 | 23 | 31 | 39 | 47 | 55 |
| 63 | 6 | 14 | 22 | 30 | 38 | 46 |
| 54 | 62 | 5 | 13 | 21 | 29 | 37 |
| 45 | 53 | 61 | 4 | 12 | 20 | 28 |

(a) Mitad izquierda

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 9 | 17 | 25 | 33 | 41 | 49 |
| 57 | 2 | 10 | 18 | 26 | 34 | 42 |
| 50 | 58 | 3 | 11 | 19 | 27 | 35 |
| 43 | 51 | 59 | 36 | 44 | 52 | 60 |

(b) Mitad derecha

**Tabla A.4:** Funcionamiento de la $PC1$ del DES

| | | | | | |
|---|---|---|---|---|---|
| 42 | 39 | 45 | 32 | 55 | 51 |
| 53 | 28 | 41 | 50 | 35 | 46 |
| 33 | 37 | 44 | 52 | 30 | 48 |
| 40 | 49 | 29 | 36 | 43 | 54 |
| 15 | 4 | 25 | 19 | 9 | 1 |
| 26 | 16 | 5 | 11 | 23 | 8 |
| 12 | 7 | 17 | 0 | 22 | 3 |
| 10 | 14 | 6 | 20 | 27 | 24 |

**Tabla A.5:** Funcionamiento de la $PC2$ del DES

| Ronda | Nº de bits |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |
| 16 | 1 |

**Tabla A.6:** Número de bits a rotar en el *key schedule* según la ronda

# B

# Funcionamiento del AES

En este anexo expondremos el funcionamiento interno del AES. Al diferencia que el DES, este algoritmo de cifrado está basando en la estructura de una **red de permutación-sustitución** o **SPN** (*Substitution-Permutation Network*) que se indica en la figura B.1. Sin embargo, ambas estructuras (SPN y Red de Feistel) comparten la idea de la aplicación de cierta operación (ronda) de manera iterativa.

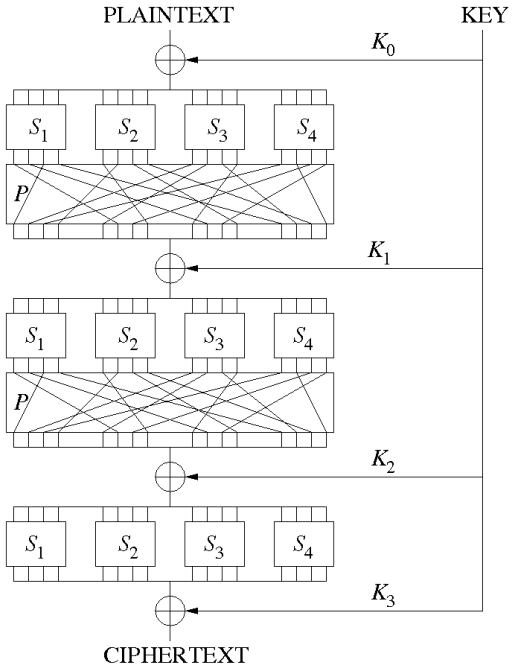



**Figura B.1:** Estructura de una red de sustitución-permutación [1]

El AES tiene un tamaño de bloque de 128 bits, pudiendo elegir como tamaño de clave 128, 192 o 256 bits. En función de ese tamaño de clave el número de rondas a aplicar es de 10, 12 y 14, respectivamente.

Este algoritmo aplica las operaciones de cada ronda sobre una structura conocida como *state*, una matriz de 4 x 4 en la que cada elemento es un byte.

[1] Extraido de https://en.wikipedia.org/wiki/Substitution-permutation_network

Cada ronda del AES se implementa combinando las siguientes 4 funciones:

- *SubBytes*: Se realiza una transformación no lineal a cada uno de los 16 bytes del *state*, viendo dichos bytes como polinomios en $GF(2^8)$ con polinomio irreducible $x^8 + x^4 + x^3 + x + 1$. En la practica se usa una tabla (B.1) precalculada con los resultados de esta operación para cada uno de los posibles valores de un byte. Por ejemplo, si introducimos el byte $34_x$, al ser sus 4 primeros bits $3_x$ (cuarta fila de la tabla) y sus 4 últimos bits $4_x$ (quinta columna de la tabla), la salida de esta operación será $28_x$. Esta operación es la que proporciona **no linealidad** al algoritmo de cifrado.
- *ShiftRows*: Se realiza una rotación hacia la izquierda a cada fila del *State*, dejando la primera fila intacta, rotando la segunda 1 byte hacia la izquierda, la tercera 2 bytes y la cuarta 3 bytes. Se puede ver visualmente este comportamiento en la ecuación B.1.
- *MixColumns*: Se multiplica cada una de las columnas del *state* por una matriz de bytes determinada, interpretando otra vez los bytes como polinomios en $GF(2^8)$. Hemos reflejado la operación que se aplica a cada columna en la ecuación B.2.
- *AddRoundKey*: Cada subclave de 128 bits obtenida aplicando el *key schedule* del AES se estructura de la misma manera que un *state* y se realiza la operación XOR entre cada byte del *state* y su correspondiente byte de la subclave.

El algoritmo de cifrado del AES comienza con una operación *AddRoundKey* tras lo cual se suceden una serie de rondas (9, 11 o 13, según el tamaño de clave escogido) que constan de la concatenación de las funciones *SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*, finalizando con una última ronda algo diferente, en la que se omite la función *MixColumns*.

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0y | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| 1y | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| 2y | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| 3y | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| 4y | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| 5y | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| 6y | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| 7y | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| 8y | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| 9y | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| Ay | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| By | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| Cy | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| Dy | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| Ey | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| Fy | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

**Tabla B.1:** Tabla de sustitución de la función *SubBytes*

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \xrightarrow{ShiftRows} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix} \tag{B.1}$$

$$\begin{pmatrix} b_{0i} \\ b_{1i} \\ b_{2i} \\ b_{3i} \end{pmatrix} = \begin{pmatrix} 2_x & 3_x & 1_x & 1_x \\ 1_x & 2_x & 3_x & 1_x \\ 1_x & 1_x & 2_x & 3_x \\ 3_x & 1_x & 1_x & 2_x \end{pmatrix} \begin{pmatrix} a_{0i} \\ a_{1i} \\ a_{2i} \\ a_{3i} \end{pmatrix} \tag{B.2}$$

La operación de descifrado, al contrario que el DES, es distinta, utilizando las funciones inversas de las ya mencionadas:

- La inversa de la función *SubBytes* consiste en realizar la sustitución opuesta a la de la tabla B.1. Por ejemplo, si introducimos el byte $4C_x$, ya que se encuentra en la sexta fila $(5y)$ y en la decimocuarta columna $(xD)$, nos devolverá el byte $5D_x$.
- La inversa de la función *ShiftRows* consiste en realizar una rotación a la derecha de cada fila del *state* de la misma cantidad de bytes que la correspondiente en la función directa.
- La inversa de la función *MixColumns* consiste en multiplicar el *state* por la matriz inversa de la correspondiente en la función directa, siendo esta matriz la de la ecuación B.3.
- La inversa de la función *AddRoundKey* es ella misma.

$$\begin{pmatrix} E_x & B_x & D_x & 9_x \\ 9_x & E_x & B_x & D_x \\ D_x & 9_x & E_x & B_x \\ B_x & D_x & 9_x & E_x \end{pmatrix} \tag{B.3}$$

El *key schedule* del AES consiste en:

- Se divide la clave original en conjuntos $K_i$ de 32 bits (4 bytes, una columna del *state*), obteniendo 4 en caso de que la clave sea de 128 bits, 6 en el caso de que la clave sea de 192 bits y 8 en el caso de que la clave sea de 256 bits.
- Se calculan los valores de una variable auxiliar *Rcon* siguiendo la recurrencia indicada por la ecuación B.4 necesarios para el cálculo de los nuevos conjuntos de 32 bits que conformarán las subclaves.
- Se generan los conjuntos de 32 bits necesarios para formar las subclaves suficientes (cada subclave necesita 4 de estos conjuntos) para las rondas del algoritmo siguiendo la recurrencia indicada por la ecuación B.5, donde $N$ es el número de conjuntos de 32 bits de la clave inicial (4, 6 u 8 en caso de que la clave sea de 128, 192 o 256 bits, respectivamente), $K_0, \ldots, K_{N-1}$ son esos grupos de bits, $R$ es el número de subclaves necesarias (11, 13 o 15 según si la clave es de 128, 192 o 256 bits, respectivamente), $i$ irá desde 0 a $4R-1$, $SW$ es la función que sustituye cada byte de la palabra de 32 bits recibida por el indicado en la tabla B.1 y $RW$ es una rotación de un byte hacia la izquierda de la palabra de 32 bits recibida.
- Las subclaves serán los *states* generados tomando grupos ordenados de 4 de los $W_i$ organizados como colum-

nas.

$$Rcon_i = \begin{pmatrix} rc_i & 0_x & 0_x & 0_x \end{pmatrix}$$

$$rc_i = \begin{cases} 1 & si \quad i = 1 \\ 2rc_{i-1} & si \quad i > 1 \quad y \quad rc_{i-1} < 80_x \\ (2rc_{i-1}) \oplus 11B_x & si \quad i > 1 \quad y \quad rc_{i-1} \geq 80_x \end{cases} \tag{B.4}$$

$$W_i = \begin{cases} K_i & si \quad i < N \\ W_{i-1} \oplus SW(RW(W_{i-1})) \oplus Rcon_{i/N} & si \quad i \geq N \quad y \quad i \equiv 0 \mod N \\ W_{i-N} \oplus SW(W_{i-1}) & si \quad i \geq N,\ N \geq 6 \quad y \quad i \equiv 4 \mod N \\ W_{i-N} \oplus W_{i-1} & en\ otro\ caso \end{cases} \tag{B.5}$$

# C

# Librerías utilizadas

## C.1. Keras y Tensorflow

Keras es una API de alto nivel que proporciona las herramientas necesarias para implementar modelos de aprendizaje automático y aprendizaje profundo de una manera sencilla y rápida, y está diseñada específicamente para la implementación de todo tipo de redes neuronales artificiales. Esta API se usa en combinación con otros frameworks de aprendizaje automático como pueden ser Tensorflow o Theano. En nuestro caso hemos escogido el uso combinado de Keras con Tensorflow, ya que nos ofrece una interfaz en la que podemos plantear numerosos experimentos en muy pocas líneas de código.

Además Keras permite una integración con el modelo CUDA, lo que nos permite acelerar de manera muy significativa los tiempos de computación por medio del uso de GPUs. Haremos uso de la GPU del autor (NVIDIA GeForce GTX 1060 6GB) para poder entrenar en tiempos razonables diferentes modelos de redes neuronales.

## C.2. Numpy y Scikit-learn

*Numpy* es una librería de Python que nos ofrece, entre otras cosas, una manejo muy eficiente de la memoria (lo que nos es especialmente útil ya que en ciertos momentos tendremos que trabajar con una cantidad significativa de memoria), por lo que lo usaremos para cargar los datos de los conjuntos de prueba en los experimentos con aprendizaje automático de una manera rápida y sencilla.

*Scikit-learn* es una librería de Python que proporciona herramientas de aprendizaje automático y aprendizaje profundo, aunque no soporta la integración con el uso de una GPU. Por ello hemos decidido usar Keras, aunque haremos uso de algunas funciones de Scikit para mostrar de manera más sencilla los resultados obtenidos tras haber realizado el entrenamiento de las redes neuronales con Keras.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID